VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Detekce chodců pomocí dronů

Pedestrian Detection Using Unmanned Aerial Vehicles

Jakub Bystričan

Bakalárska práca

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

Ostrava, 2021

Abstrakt

V tejto práci boli porovnávané metódy detekcie chodcov na snímkoch z dronov pomocou konvolučných neurónových sietí. Boli použité dve detekčné siete - YOLOv5 a Retinanet. U týchto sietí bola porovnávaná ich presnosť, rýchlosť a náročnosť trénovania. Taktiež bol sledovaný vplyv niektorých parametrov trénovania na výsledky detekcie. Pre trénovanie a testovanie bol použitý Stanford Drone Dataset ktorý obsahuje videozáznamy zachytené s dronom z kampusu Univerzity v Stanforde.

Kľúčové slová

konvolučné neurónové siete, detekcia chodcov, hlboké učenie,YOLOv5, Retinanet

Abstract

The main focus of this study was pedestrian detection using drones and convolutional neural networks. 2 detection networks were used - YOLOv5 and Retinanet. The performance was compared based on precision and speed of detection and the demands on training process. Impact of certian training parameters on results was also observed. For training and testing Stanford Drone Dataset was used, containing video recordings captured by drones at Stanford University campus.

Keywords

convolutional neural networks, pedestrian detection, deep learning, YOLOv5, Retinanet

Obsah

Zoznam použitých skratiek a symbolov

ROI	–	Region of Interest
CNN	–	Convolutional Neural Network
HOG	–	Histogram of Oriented Gradients
SVM	–	Support Vector Machine
R-CNN	–	Region-based Convolutional Neural Network
FPN	–	Feature Pyramid Network
AP	–	Average Precision
mAP	–	mean Average Precision
YOLO	–	You Only Look Once
RFB	–	Receptive Field Block
LBP	–	Local Binary Patterns
UAV	–	Unmanned Aerial Vehicle
SDD	–	Stanford Drone Dataset
LCCNET	–	Local Context Encode Network

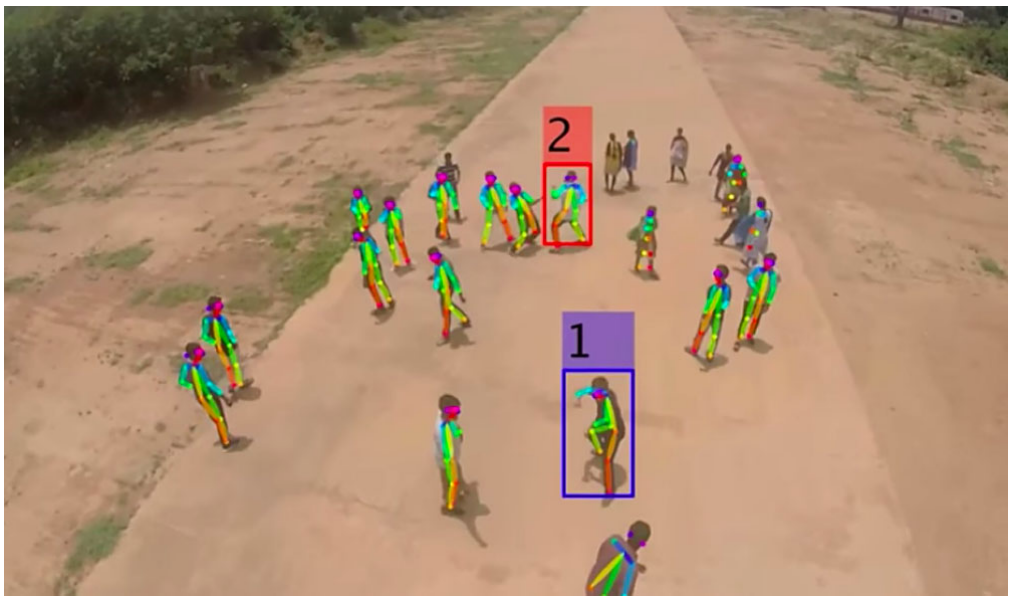
Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

V poslednej dekáde začali byť drony používané v mnohých oblastiach ako napr. záchranné misie, sledovanie ilegálnych imigrantov, monitorovanie davov pri evakuácii alebo monitorovanie dopravnej situácie. Podľa nedávnej štatistiky [1] až 1 800 tínedžerov a detí bolo nahlásených ako nezvestných. Pomocou dronou môžu byť zachytené celé hodiny záznamov. Manuálne kontrolovanie takýchto záznamov je pre človeka veľmi dlhá a náročná úloha. Automatizované a presné detekčné systémy ktoré by dokázali detekovať chodcov na snímkoch z veľkej výšky by mohli hrať významnú úlohu hlavne v sledovacích a záchranných misiách.



Obr. 1.1: Použitie detekcie chodcov pomocou dronu, pri hľadaní násilných aktivít. (prevzaté z [2])

Na obrázku 1.1 je príklad použitia detekcie chodcov pomocou dronov. Okrem samotnej detekcie prítomnosti chodcov je tu detekovaná aj ich násilná aktivita. Pri detekcii chodcov teda môže byť detekovaná podľa ich postojov aj činnosť ktorú vykonávajú. Takáto detekcia by mohla nájsť využite

pri monitorovaní bezpečnosti v mestách, prípadne odhaľovaní verejných nepokojov ale aj iných nezákonných činností.

Detekcia chodcov na takýchto snímkoch je náročná úloha. Chodci sú z veľkej výšky len ťažko rozoznateľný od iných objektov. Rôzne svetelné alebo poveternostné podmienky môžu túto úlohu ešte viac zaťažiť. V oblasti detekcie objektov dnes dominujú metódy ktoré zahŕňajú použitie konvolučných neurónových sietí. Cieľom tejto práce bolo nájsť vhodnú detekčnú sieť, ktorá by dokázala rýchlo a efektívne detekovať chodcov na snímkoch z dronov.

V prvej kapitole je rozoberaná architektúra konvolučných neurónových sietí a architektúry rôznych detekčných sietí ktoré sa dnes používajú na detekciu objektov. V kapitole 3 boli zhrnuté niektoré metódy ktoré už boli v tejto oblasti použité. V kapitole 4 sú popísané existujúce datasety, ktoré obsahujú snímky chodcov z dronov a možnosti ich použitia. V kapitole 5 bol vybratý vhodný dataset a dve detekčné siete ktoré boli prispôsobené na detekciu malých objektov niektorými technikami z kapitoly 3. U týchto sietí bol nakoniec porovnávaný proces trénovania a výsledky aké pri detekcii dosiahli.

Kapitola 2

Konvolučné neurónové siete (CNN)

Konvolučné neuronové siete (Convolutional Neural Networks – CNN) sú špeciálnym typom neurónových sietí navrhnuté hlavne pre prácu s viac-dimenzionálnymi dátami (obrázky, videá....). Pri práci s digitálnymi obrázkami poskytujú CNN niekoľko výhod v porovnaní s obyčajnými neurónovými sieťami. Vďaka svojej architektúre dokážu zredukovať počet vstupných parametrov a tým výrazne znížiť výpočetné nároky na trénovanie. Ďalšou výhodou je že vzory ktoré sa CNN učí sú priestorovo nezávislé a dokážu byť detekované nezávisle na ich polohe v obraze.

V roku 1989 navrhol Yann LeCun so svojím tímom sieť Lenet-5[3] ktorá sa stala prvým významným míľnikom vo svete CNN. Dnes sú CNN najčastejšou voľbou pri detekcii ľudí ale aj objektov ako takých. Okrem Lenet-5 už dnes existuje mnoho iných komplexnejších a úspešnejších sietí ako napr. Resnet [4], Alexnet [5] alebo VGG16 [6].

2.1 Architektúra konvolučných neurónových sietí

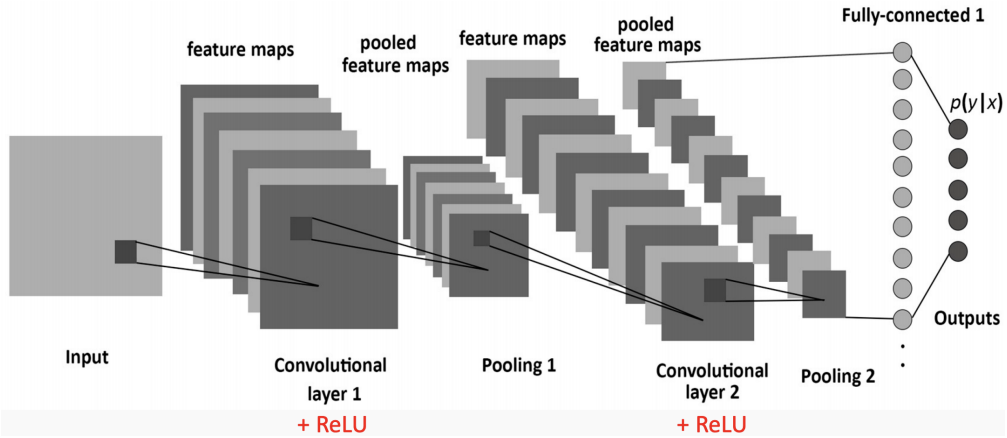
CNN sú tvorené tromi základnými typmi vrstiev. Sú to konvolučná vrstva, pooling vrstva a plne-prepojená vrstva. Architektúra CNN je tvorená viacerými týmito vrstvami ukladanými za sebou ako je to vidieť na obrázku 2.1.

Konvolučná vrstva (*convolutional layer*) je základným stavebným blokom CNN. Úlohou konvolučnej vrstvy je extrakcia príznakov. Jej názov je odvodený od matematickej operácie - konvolúcie, ktorá sa v tejto vrstve aplikuje. Nasledujúca rovnica je matematickým vyjadrením konvolúcie:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \tag{2.1}$$

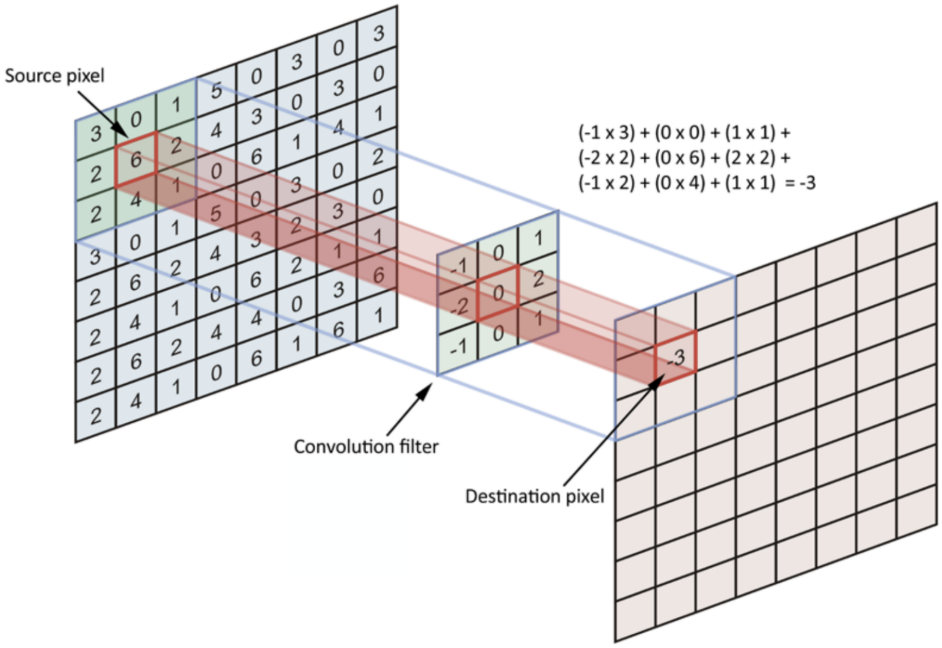
kde f označuje vstupnú maticu (obrázok) a h filter. Indexy riadkov a stĺpcov výstupnej matice sú označené ako m a n [8].

Konvolúcia teda prijíma dva vstupy. Prvým je matica pixelov reprezentujúca obrázok, druhým je filter, inak nazývaný aj konvolučné jadro (*kernel*). Filter je maticou s malými rozmermi, najčastejšie



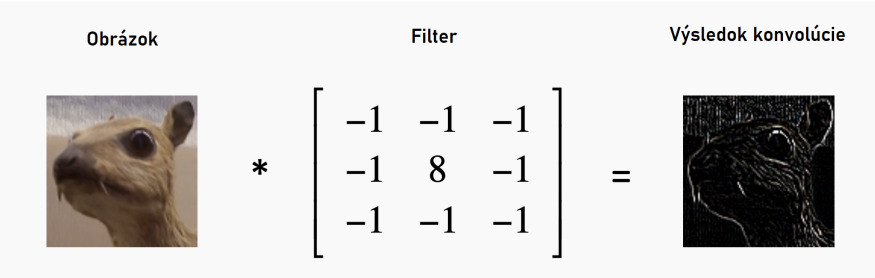
Obr. 2.1: Príklad CNN s dvoma konvolučnými vrstvami, dvoma pooling vrstvami, a plne prepojenou vrstvou. (prevzaté z [7])

2x2 alebo 3x3 ale môže mať aj iné rozmery. Filter je potom po krokoch (*stride*) s určitou veľkosťou posúvaný cez celú maticu. Výstupom konvolučnej vrstvy je tzv. príznaková mapa (*feature map*). Spôsob výpočtu konvolúcie je ilustrovaný na obrázku 2.2.



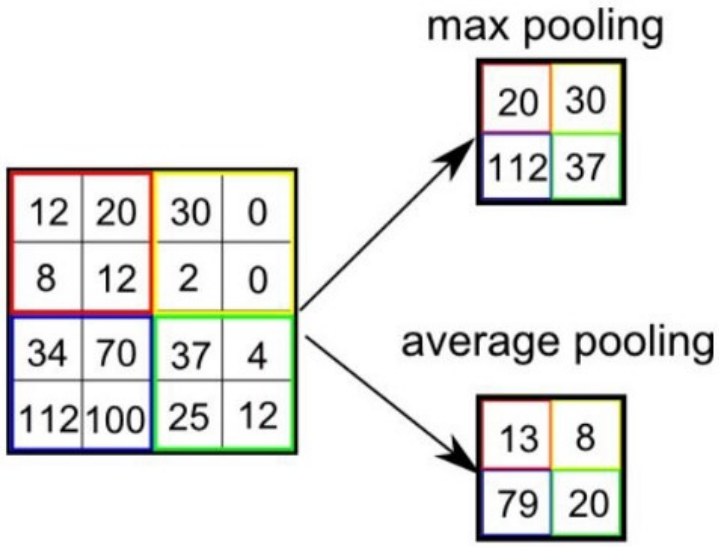
Obr. 2.2: Výpočet konvolúcie matice 8x8 s filtrom 3x3. (prevzaté z [7])

Konvolučné vrstvy používajú rôzne typy filtrov. Konvolúcia obrázka s takýmito filtrami môže predstavovať operácie ako napr. detekcia hrán, rozmazanie, zostrenie a pod. [9]. Pri procese trénovania sa práve hodnoty v týchto filtrov upravujú. Na obrázku 2.3 je príklad filtru použitého na detekciu hrán objektu.



Obr. 2.3: Príklad filtru použitého na detekciu hrán. (prevzaté z [7])

Pooling vrstva má za úlohu podvzorkovanie (*subsampling*) - zmenšenie dimenzionality príznakovej mapy a tým zmenšeniu počtu parametrov. Zníženie počtu parametrov zároveň znamená zníženie výpočetnej náročnosti. Podstatou tejto operácie je združovanie niekoľkých susedných buniek do jednej. Pooling môže pripomínať znižovanie rozlíšenia obrázka [9].



Obr. 2.4: Porovnanie:*max-pooling* a *average-pooling* (prevzaté z [7])

Najčastejšou formou poolingu je *max-pooling* kedy sa zo združovaných buniek vyberá maximálna hodnota. V praxi je často používaný *max-pooling* s filtrom 2x2 a veľkosťou kroku 2. Tým dosiahneme zmenšenie príznakovej mapy o polovicu [10]. Iným typom poolingu je *average-pooling*, kedy výstupom združovaných buniek je ich priemerná hodnota. *Max-pooling* a *average-pooling* sú znázornené na obrázku 2.4.

Plne prepojená vrstva (*fully connected layer, dense layer*) obsahuje neuróny z ktorých každý je prepojený s každým uzlom v susedných vrstvách. Toto usporiadanie je podobné umelým neurónovým sietiam a spôsobuje vyššie výpočetné nároky. Preto bývajú tieto vrstvy umiestnené až ako koncové vrstvy CNN používané na mapovanie príznakov na triedy. Výpočetné nároky týchto vrstiev

a taktiež riziko pretrénovania siete môžu byť znížené používaním tzv. *dropout* techniky [5].

2.2 Detekčné siete

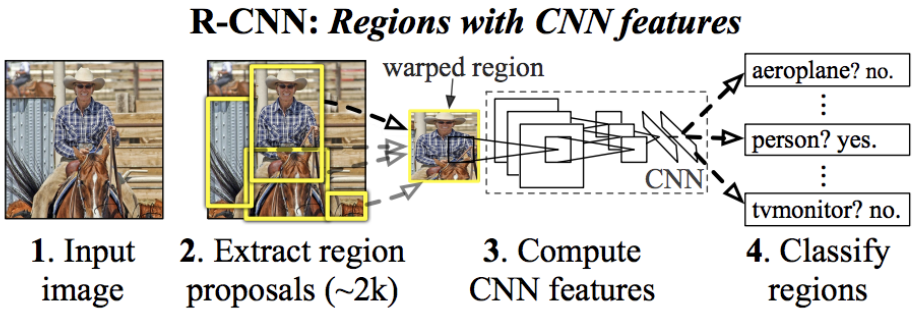
Architektúry spomenuté v úvode kapitoly 2 sú navrhnuté len pre klasifikáciu obrazu. Inými slovami, predpokladá sa že na obrázku sa vyskytuje len jeden objekt, ktorý má sieť za úlohu identifikovať. V prípade že chceme v obrázku rozoznávať viaceré objekty s rôznymi veľkosťami potrebujeme zaistiť spôsob ktorým sieť v obrázku nájde kandidátov na takýto objekt. Oblasť okolo objektu potom vyreže a odovzdá klasifikátoru ktorý objekt správne identifikuje.

Detektory môžeme rozdeliť do dvoch skupín - jednostupňové (*one-stage*) a dvojstupňové (*two-stage*). Dvojstupňové detektory v prvom kroku vygenerujú oblasti zájmu (*region of interest - ROI*) ktoré odovzdajú klasifikátoru ktorý ich v druhom kroku identifikuje. Jednostupňové detektory vykonávajú detekciu aj klasifikáciu v jednom kroku. Jednostupňové detektory sú obyčajne rýchlejšie v porovnaní s dvojstupňovými, dosahujú však nižšiu presnosť [11].

2.2.1 R-CNN

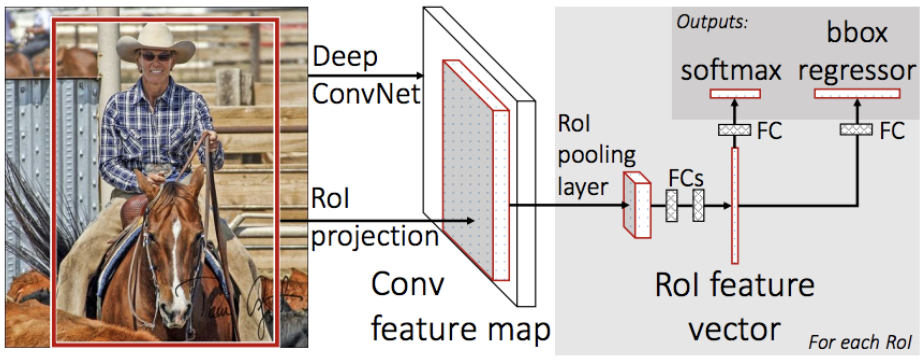
R-CNN a jej ďalšie modifikácie (*Fast R-CNN, Faster R-CNN, Mask R-CNN*) sú dvojstupňový detektory, ktoré znamenali pokrok v detekcii objektov.

R-CNN na hľadanie oblastí záujmu v obrázku používa algoritmus zvaný Selektívne vyhľadávanie popísaný v práci [12]. Tento algoritmus prechádza obraz oknami rôznych veľkostí pričom sa snaží spájať pixely s podobnou farbou alebo intenzitou, za predpokladu že by mohli náležať rovnakému objektu. Týmto spôsobom R-CNN vygeneruje približne 2000 oblastí ktoré by mohli obsahovať nejaký objekt (*region proposals*) ako je to znázornené na obrázku 2.5. Tieto oblasti sa potom zde-



Obr. 2.5: Schéma fungovania R-CNN. (prevzaté z [13])

formujú a odovzdajú sa konvolučnej neurónovej sieti Alexnet [5]. Alexnet v tomto prípade neslúži na konečnú klasifikáciu objektu ale len na extrakciu príznakov. Výstupom je 4096-dimenzionálny vektor príznakov, na základe ktorého potom SVM klasifikátor identifikuje objekt [13].



Obr. 2.6: Schéma fungovania Fast R-CNN. (prevzaté z [13])

Napriek tomu že R-CNN bol významným pokrokom v oblasti detekcie objektov, trénovanie takéhoto modelu bolo časovo náročné keďže pre každý obraz musela sieť klasifikovať približne 2000 oblastí. Ďalšou nevýhodou bola časová náročnosť samotnej detekcie a nebola teda možná v reálnom čase.

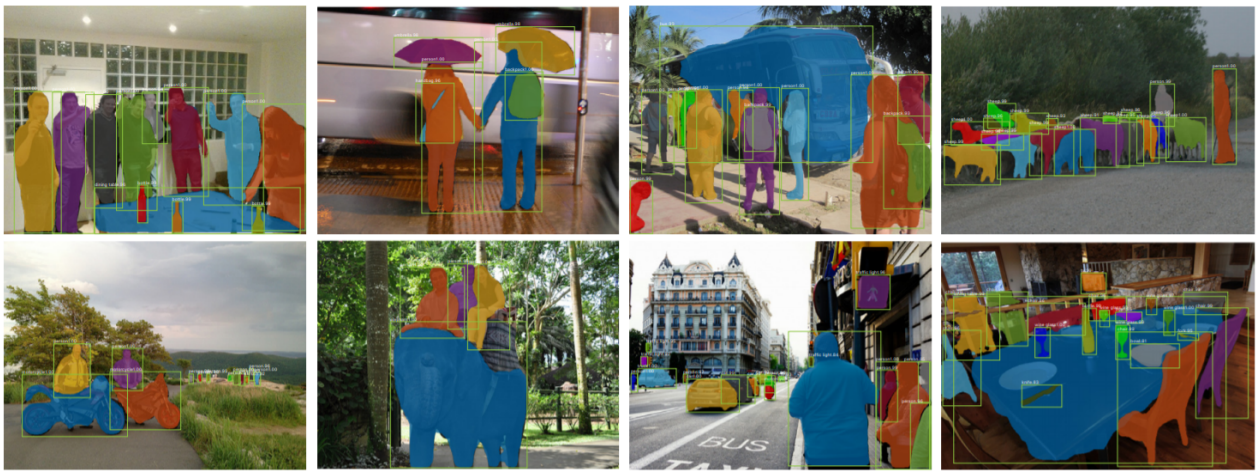
Autori pôvodnej R-CNN sa rozhodli tieto nedostatky riešiť a vytvorili nový detektor nazývaný **Fast R-CNN**. Jednou zo zmien oproti pôvodnému modelu bola použitá konvolučná sieť - VGG16 [6]. Podstatnejšou zmenou však bolo, že obraz je najprv odovzdaný konvolučnej neurónovej sieti (viz. obr. 2.6), ktorá vygeneruje mapu príznakov a až potom sú vybrané oblasti záujmu. Sieť už teda nemusí spracovávať približne 2000 obrazov (jeden pre každú navrhnutú oblasť záujmu) čo výrazne urýchlilo proces detekcie aj trénovania [14].

Obidva vyššie spomenuté detektory (R-CNN a Fast R-CNN) používali pre navrhovanie oblastí záujmu Selektívne vyhľadávanie[12], čo je časovo náročný proces ovplyvňujúci výkon siete. Ďalšou modifikáciu R-CNN bola **Faster R-CNN** ktorá tento problém eliminovala. Jej autori použili pre navrhovanie oblastí záujmu samostatnú konvolučnú neurónovú sieť a nazvali ju *Region Proposal Network (RPN)*. Tento prístup výrazne zefektívnil proces navrhovania oblastí záujmu a tým zvýšil rýchlosť detekcie do takej miery že umožnil detekciu v reálnom čase [15].

Najnovšou modifikáciou R-CNN je **Mask R-CNN**[16] vyvinutá výskumným tímom z Facebook AI. Faster R-CNN má pre každého kandidáta na objekt dva výstupy: triedu (názov objektu) a súradnice ohraničujúce objekt. Mask R-CNN pridáva tretí výstup a tým je maska objektu - identifikuje každý pixel, ktorý objektu patrí. Príklady detekcie objektov Mask R-CNN vrátane masky sú na obrázku 2.7.

2.2.2 Retinanet

Retinanet patrí k jednostupňovým detektorom, napriek tomu sa svojou presnosťou vyrovná aj dvojstupňovým. Vyššiu presnosť v porovnaní s inými jednostupňovými detektormi dosahuje hlavne



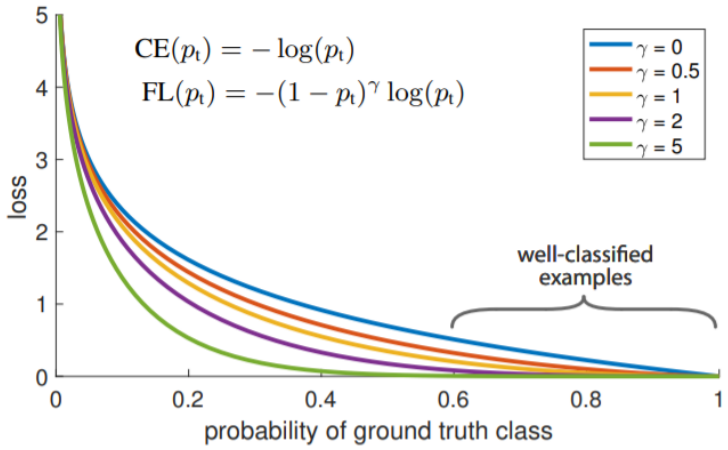
Obr. 2.7: Výsledky Mask R-CNN na testovacom datasete COCO. (prevzaté z [16])

vďaka novej chybovej funkcii **Focal Loss** a modulu nazvanému **Feature Pyramid Network (FPN)** [17].

Autori modelu Retinanet sa snažili objasniť dôvody, prečo jednostupňové detektory dosahujú nižšiu presnosť ako dvojstupňové. Ako hlavný dôvod označili vysokú nevyrovnanosť tried pozadia a samotných objektov (*foreground-background class imblance*) počas trénovania. Väčšina jednostupňových detektorov totiž spracováva veľké množstvo kandidátov na objekt (10 000 - 100 000) ale len malé množstvo skutočne obsahuje nejaký objekt. Väčšina týchto kandidátov je ľahko klasifikovaná ako pozadie, resp. žiaden objekt. To ale negatívne vplýva na proces učenia, pretože celková strata na týchto jednoduchých klasifikáciách je vo výsledku oveľa vyššia ako strata na klasifikáciách ktoré objekt obsahujú a preto majú tieto jednoduché klasifikácie vyšší vplyv na proces učenia. To môže viesť až degenerácii modelu. Dvojstupňové detektory ako napríklad Faster R-CNN tento problém riešia už vo fáze návrhu kandidátov na objekt kde sa počet kandidátov zníži na 1000 - 2000 a a odfiltruje sa väčšina vzoriek obsahujúcich len pozadie [17].

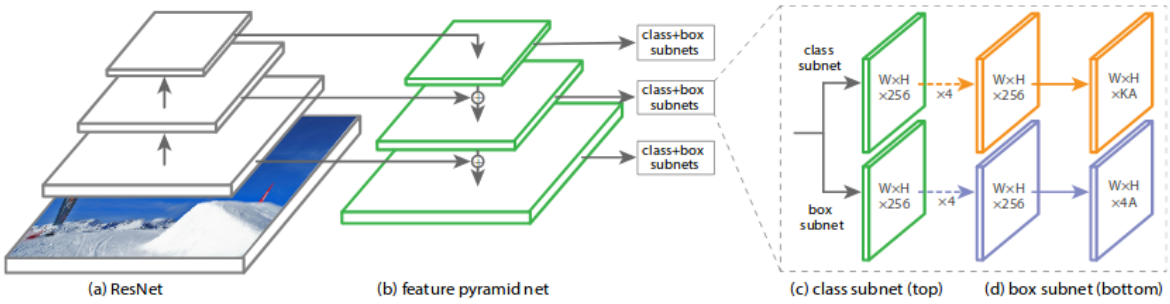
Preto autori prišli s novou chybovou funkciou nazvanou **Focal Loss** ktorá sa postará o túto nevyrovnanosť tried a zníži vplyv negatívnych vzoriek (vzoriek neobsahujúcich objekt) na proces trénovania. Focal Loss je založená na chybovej funkcii *Cross Entropy* o ktorej sa môže čitateľ v prípade záujmu dozvedieť viac v článku [18]. Focal Loss obsahuje naviac parameter γ ktorý nadobúda kladné hodnoty. Čím je jeho hodnota vyššia, tým viac detektor preferuje učenie z ťažkých dát (vzoriek obsahujúcich objekt). Autori uvádzajú že v praxi dosahuje model najlepšie výsledky pri hodnote $\gamma = 2$ [17]. V prípade že sa $\gamma = 0$ je funkcia ekvivalentná s funkciou Cross Entropy [17]. Na obrázku 2.8 je zobrazený vplyv parametru γ na výslednú chybu.

Feature Pyramid Network je modulom rozširujúcim štandardnú konvolučnú sieť. Tento modul efektívne vytvára pyramídu príznakových máp pričom používa výstupy konvolučných vrstiev z konvolučnej siete. Pyramída teda obsahuje mapy príznakov v rôznych rozlíšeniach. Týmto spôsobom



Obr. 2.8: Graf ktorý zachytáva vplyv parametru γ na výslednú chybu. (prevzaté z [17])

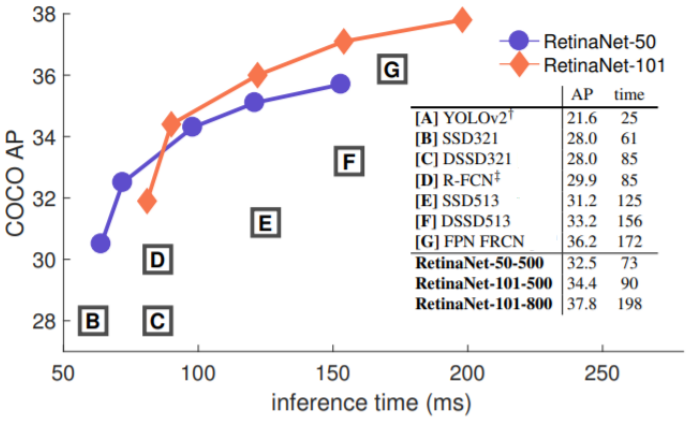
dokáže získať slabé príznaky z nízkych rozlíšení a zároveň silné príznaky z vyšších rozlíšení. Každá vrstva pyramídy potom môže byť byť použitá na detekciu objektov v rôznych veľkostiach [17].



Obr. 2.9: Architektúra Retinanet. (prevzaté z [17])

Architektúru detektoru Retinanet môžme do štyroch hlavných komponentov znázornených na obrázku 2.9. Prvým je konvolučná sieť ktorá slúži na získanie príznakových máp. Autori modelu použili sieť Resnet od spoločnosti Microsoft [4], je však možné zvoliť aj iný model. Resnet poskytuje viaceré variácie s rôznou hĺbkou, resp. počtom hlbokých vrstiev (*napr. Resnet-50, Resnet-101*). Výstupy konvolučných vrstiev sú použité druhým komponentom - Feature Pyramid Network. Zvyšné dva komponenty sú podsiete nazvané *class subnet* a *box subnet*. *Class subnet* je zodpovedná za klasifikáciu objektu, zatiaľ čo *box subnet* slúži na regresiu ohraničení objektov [17].

Autori Retinanet porovnávali rýchlosť a presnosť detekcie s inými jednostupňovými detektormi na testovacom datasete COCO. Výsledky sú zobrazené na obrázku 2.10. Porovnané boli tri variácie Retinanet s použitím siete Resnet-50 aj Resnet-101.



Obr. 2.10: Porovnanie Retinanet s inými jednostupňovými detektormi s hľadiska presnosti AP (average precision) a času ptrebného na detekciu.(prevzaté z [17])

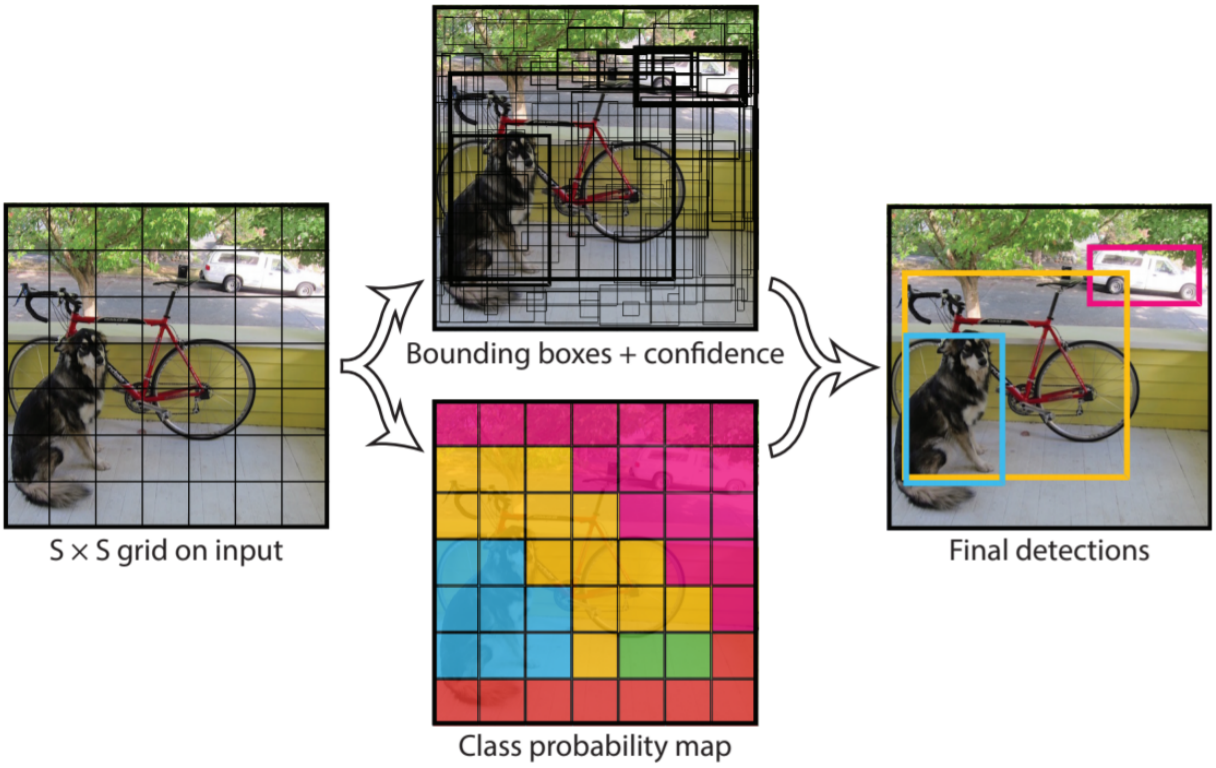
2.2.3 YOLO

YOLO (*You Only Look Once*) je detekčný algoritmus uvedený v roku 2016. Populárnym sa stal predovšetkým vďaka svojej rýchlosti. V tomto ohľade dosahoval výrazne lepšie výsledky ako dovtedy známe detekčné algoritmy. Aj dnes je YOLO (jeho novšie verzie) častou voľbou pri detekcii v reálnom čase.

Detektory ako napr. Fast R-CNN detekujú objekty v obraze v dvoch krokoch. V prvom sa vytvoria návrhy oblastí, ktoré by mohli obsahovať nejaký objekt a v druhom sú tieto oblasti odovzdané klasifikátoru, ktorý ich identifikuje. YOLO sa však na obrázok pozrie iba raz a tieto kroky prebiehajú v jednej fáze pomocou jednej konvolučnej siete. Celý proces je preto jednoduchší a rýchlejší [19].

YOLOv1 bolo prvou verziou. Autori tu predstavili svoj algoritmus, ktorý spočíval v rozdelení obrazu na mriežku o veľkosti SxS rovnakých buniek, najčastejšie 19x19. Pre každú bunku je potom navrhnutý istý počet ohraničujúcich boxov (*bounding box*) a vypočíta sa dôveryhodnostné skóre (*confidence score*) - ako veľmi si je box istý že nejaký objekt obsahuje, nie je však dôležité aký objekt to je. Ďalej sa pre každú bunku vypočíta pravdepodobnosť s akou táto bunka objekty obsahuje. Vyberie sa pravdepodobnosť s najvyššou hodnotou a tento objekt sa priradí bunke. Týmto spôsobom vznikne mapa pravdepodobností tried (viz. obr. 2.11). Pravdepodobnosti tried sa potom vynásobia s dôveryhodnostným skóre jednotlivých ohraničujúcich boxov čím dostaneme dôveryhodnostné skóre pre špecifický objekt (*class-specific confidence score*). Na základe tohto skóre môže zahodiť niektoré ohraničujúce boxy pretože ich skóre je nízke a to znamená že pravdepodobne objekt neobsahujú alebo ho ohraničujú nesprávne. Pre jeden objekt tak ale môže stále ostávať viac ako jeden ohraničujúci box. Preto sa na tieto boxy aplikuje potlačenie nemaxím (*non-maximal suppression*), ktoré by nás malo týchto duplicít zbaviť [19].

Napriek svojej rýchlosti mal stále YOLOv1 nedostatky. Jeden z nich je obmedzené množstvo



Obr. 2.11: Algorimus YOLO.(prevzaté z [19])

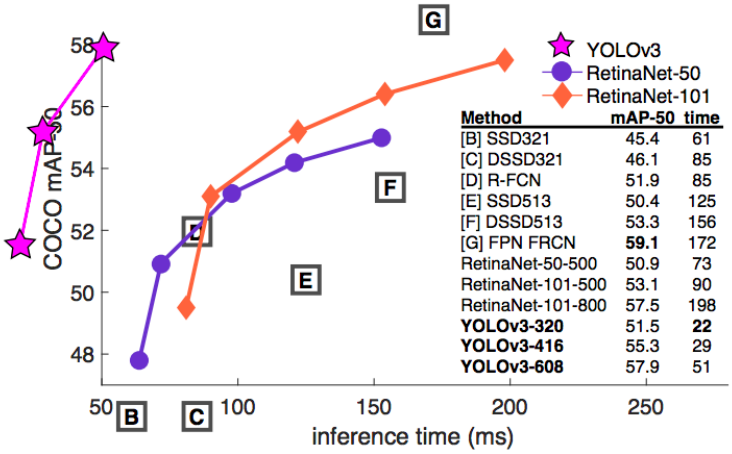
tried ktoré môže detekovať. Napr. pri rozdelení obrázku na mriežku 19x19 buniek, počet možných tried je $19 * 19 = 361$. Ďalším nedostatkom bola aj nízka presnosť obzvlášť pri malých objektoch. **YOLOv2** malo niektoré z vyššie spomenutých nedostatkov riešiť. Novinkou bola napríklad normalizácia dávok (*batch normalization*). To zvýšilo presnosť (mAP) o 2% a zároveň mohli autori vynechať dropout techniku bez rizika pretrénovania modelu. Pôvodný YOLO model začínal trénovanie na obrázkoch s rozlíšením 224x224 a pri detekcii zvýšil rozlíšenie na 448x448. YOLOv2 používa pri trénovaní v prvých 10 epochách rozlíšenie 448x448. To viedlo k zvýšení presnosti o ďalšie 4%. V prvej verzie boli ohraničujúce boxy predikované pomocou plne prepojených vrstiev. Tie boli odstránené a namiesto nich bola použitá konvolučná vrstva s tzv. kotviacimi boxmi (*anchor boxes*). Narozdiel od Fast R-CNN, tieto boxy nie sú pevne preddefinované ale boli získané pomocou zhlukovej analýzy (*k-means clustering*) na ohraničujúcich boxoch z datasetu COCO a Pascal VOC. Pre každú bunku v mriežke je potom vygenerovaných 5 kotviacich boxov čo je podľa autorov ideálny pomer medzi presnosťou a komplexnosťou modelu [20].

V roku 2018 prišli rovnakí autori s novou verziou - **YOLOv3**. Je tu použitá nová sieť nazvaná Darknet-53 vychádzajúca zo siete Resnet. Autori však uvádzajú že Darknet-53 je presnejšia a až 1,5 krát rýchlejšia ako Resnet-101. Ako napovedá jej názov obsahuje 53 vrstiev čo je v porovnaní s 19

Detection Frameworks	Train	mAP	FPS
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG-16	2007+2012	73.2	7
Faster R-CNN ResNet	2007+2012	76.4	5
YOLO	2007+2012	63.4	45
SSD300	2007+2012	74.3	46
SSD500	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Obr. 2.12: Porovnanie presnosti a rýchlosti YOLOv2 s inými detektormi.(prevzaté z [19])

vrstvami v YOLOv2 takmer dvojnásobok. To má za následok mierne zníženie rýchlosti, avšak vyššiu presnosť. YOLOv3 naviac detekuje objekt až v troch vrstvách (v troch rôznych rozlíšeniach). Vrstva 13x13 je zodpovedná za detekciu veľkých objektov, vrstva 26x26 za detekciu stredných objektov a vrstva 52x52 za detekciu malých objektov. To výrazne zlepšilo detekciu malých objektov, čo je jedným z nedostatkov algoritmu YOLO. Ďalšou novinkou v YOLOv3 je tzv. multi-label klasifikácia tzn. že jeden objekt môže spadať do viacerých tried napr. do tried človek a dieťa. [21]

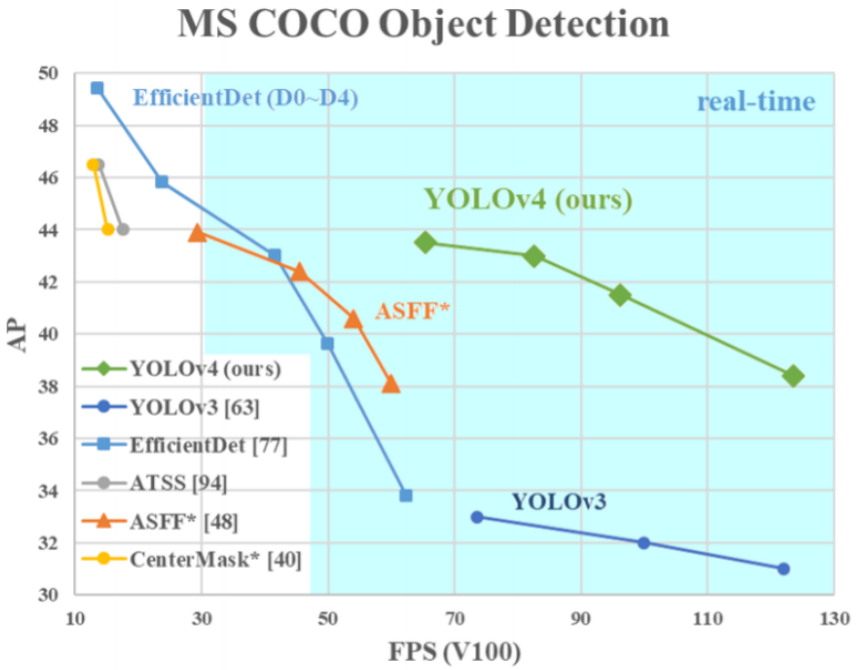


Obr. 2.13: Porovnanie presnosti a rýchlosti YOLOv3 s inými detektormi.(prevzaté z [21])

V roku 2020 bola uvedená verzia **YOLOv4** tento krát od iného autora - Alexaya Bochkovskiy. Pôvodný autor algorimu YOLO - Joseph Redmon sa prestal podieľať na vývoji YOLO a aplikácii počítačového videnia všeobecne, kvôli etickým dôvodom. Na svojom twtittery uviedol, že má obavy o zneužitie týchto technológií pre militaristické aplikácie a narušenie súkromia [22].

YOLOv4 opäť prináša zvýšenie presnosti a rýchlosti. Jeho autori vo svojej práci zahrnuli tzv.

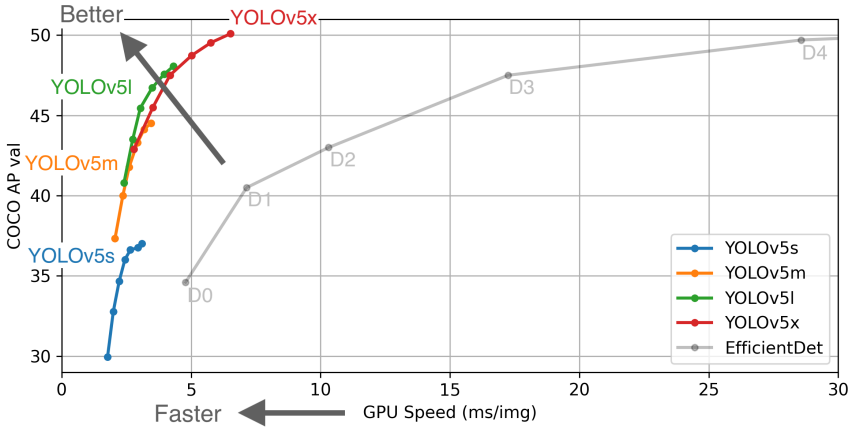
bag of freebies a *bag of specials*. *Bag of freebies* obsahuje techniky používané počas trénovania, ktoré zvyšujú presnosť detekcie ale neznížia jej rýchlosť. Príkladom takejto techniky rozšírenie dát (*data augmentation*), kedy sú zo vstupných obrazov vytvorené nové obrazy rozmazaním, otočením a pod. čím sa zvýši robustnosť detekcie. *Bag of specials* naproti tomu zahŕňa metódy ktoré v malej miere znížia rýchlosť detekcie ale zvýšia jej presnosť. Príkladom takejto metódy je zväčšenie *receptive field*. V porovnaní s verziou 3 dosiahlo YOLOv4 zvýšenie výkonu až o 12% v rýchlosti a až 10% v presnosti. Na obrázku 2.14 môžme vidieť porovnanie s inými detektormi vrátane YOLOv3 a významného konkurenta YOLO - EfficientDet. Tieto výsledky boli merané na grafickej karte NVIDIA Tesla V100 s datasetom MS COCO. [23].



Obr. 2.14: Porovnanie presnosti a rýchlosti YOLOv4 s inými detektormi.(prevzaté z [23])

Len o dva mesiace neskôr predstavila spoločnosť Ultralytics **YOLOv5**. Napriek tomu že je táto verzia považovaná za neoficiálnu verziu YOLO, má mnoho výhod a určite stojí za zváženie. O tom či je výkonnejšia v porovnaní s verziou 4 sa stále vedú diskusie, niektoré zdroje však uvádzajú že dosahuje vyššiu presnosť aj rýchlosť [24]. Hlavnou zmenou je použitý framework PyTorch namiesto frameworku Darknet v predchádzajúcich verziách. PyTorch má oproti Darknetu väčšiu komunitu uživateľov a uľahčuje nasadzovanie do produkčného prostredia. V predchádzajúcich verziách existovali len dve varianty algoritmu - plnohodnotná verzia YOLO a tiny-Yolo s nižšími výpočetnými nárokmi používané hlavne na mobilných a embedded zariadeniach. YOLOv5 poskytuje až 4 varianty: `s, m, l, x`. Varianta `s` má najnižšie výpočetné nároky avšak za cenu nižšej presnosti. Varianta `x` má naopak nároky najvyššie a poskytuje aj najvyššiu presnosť detekcie [25]. Porovnanie týchto

variánt môžeme vidieť na obrázku 2.15.



Obr. 2.15: Porovnanie 4 variánt YOLOv5 a konkurenčnej EfficientDet.(prevzaté z [25])

Kapitola 3

Existujúce riešenia

V tejto kapitole uvádzam niektoré metódy ktoré boli použité na detekciu chodcov pomocou dronov. V oblasti detekcie objektov a aj počítačového videnia vo všeobecnosti v súčasnosti dominujú konvolučné neurónové siete. Väčšina metód popísaných v tejto kapitole zahŕňa použitie detekčných sietí z podkapitoly 2.2. Chodci na snímkoch z výšky sú malý a často veľmi ťažko rozlíšiteľný od iných objektov. Nižšie uvedené metódy teda zahŕňajú rôzne modifikácie existujúcich detekčných sietí pre zvýšenie presnosti detekcie malých objektov.

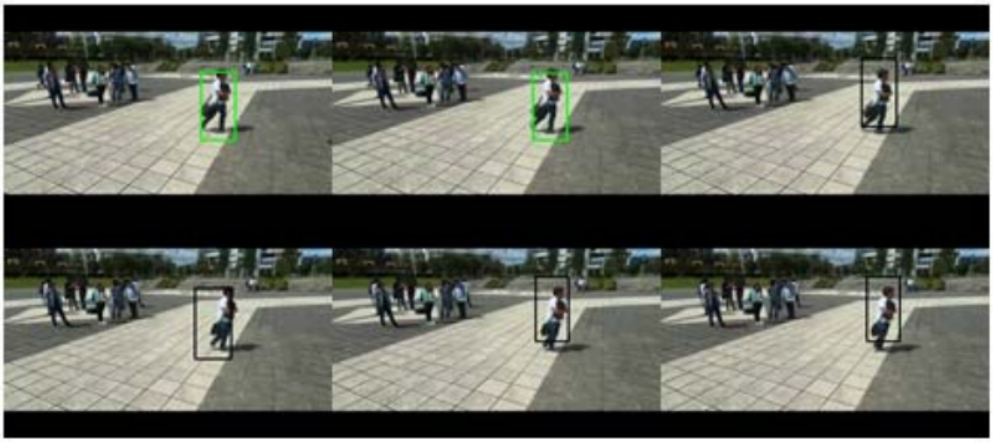
3.1 Haar-LBP príznaky + Adaboost

Metóda použitá v práci [26] je jedinou metódou v tejto kapitole ktorá nezahŕňa použitie konvolučných neurónových sietí. Autori tejto práce sa pokúsili detekovať chodcov na snímkoch z dronov kombináciou niekoľkých algoritmov. Pre získavanie príznakov bola použitá kombinácia Haar-Like Features a Local Binary Patterns (ďalej už len LBP) [26].

LBP je deskriptor textúry používaný pre deteckiu objektov. Jeho princíp spočíva v porovnávaní centrálneho pixelu s jeho susedmi. Algoritmus Haar-Like Features používa na získavanie príznakov zmeny intenzity pixelov v určitých regiónoch obrazu. Pri detekcii objektov je potom obraz skenovaný špecifickými oknom určitej veľkosti obsahujúcim tieto špecifické Haar-Like príznaky [26].

Pre klasifikáciu bola použité technológia Adaboost (Adaptive Boost). Je to algoritmus strojového učenia založená na tzv. "boostingu". Táto metóda vytvára veľmi presný a nelineárny klasifikátor z viacerých pomerne slabých a nepresných lineárnych klasifikátorov (*weak kernels*). Pre zvýšenie výkonu klasifikátoru implementovali autori Meanshift algoritmus ktorý sleduje príznaky detekcie [26].

Dataset na ktorom bola táto metóda testovaná obsahoval približne 5400 obrázkov. Na rovnakom datasete autori otestovali aj metódu s použitím HOG(*Histogram of oriented gradients*). Testovanie bolo rozdelené na tri časti podľa výšky z akej boli snímky zachytené - 2 m, 3 m a 4 m. Metóda HAAR-

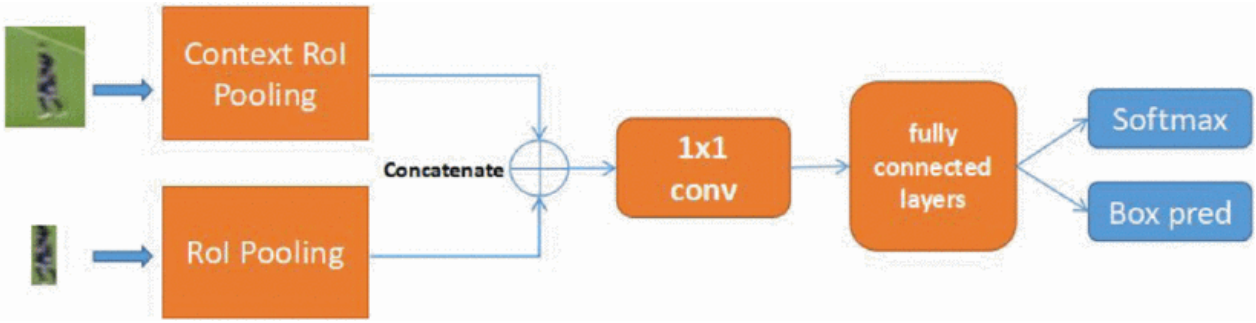


Obr. 3.1: Výsledky detekcie bez použitia Meanshift (čierny box) a s použitím Meanshift (zelený box) [26].(prevzaté z [26])

LBP nebola schopná v presnosti detekcie prekonať metódu HOG. Autori práce však preukázali že použitie algoritmu Meanshift ju dokázalo značne zvýšiť ako je vidieť na obrázku 3.1.

3.2 Faster R-CNN + LCCNET

Autori práce [27] použili upravený algoritmus Faster R-CNN na detekciu chodcov v snímkoch z dronov. Detekcia chodcov na takýchto snímkoch je náročná pretože z cieľového objektu sa dá získať len málo informácií. Preto autori v práci využívajú aj okolie objektu, ktoré nazvali kontext. Autori vylepšili pôvodnú RPN a nazvali ju Inception-RPN. Okrem kandidáta na objekt vygeneruje aj kandidáta s jeho kontextom. Obaja kandidáti sú potom odovzdaný ďalšiemu modulu nazvanému LCCNET (Local Context Encode Network). LCCNET obsahuje dve ROI pooling vrstvy - jednu pre kontext a jednu pre samotný objekt. Ich výsledné mapy príznakov sa potom spájajú a odovzdávajú konvolučnej vrstve ktorá ich zlúči a odovzdá na klasifikáciu plne prepojenej vrstve.



Obr. 3.2: Schéma fungovania LCCNET. (prevzaté z [27])

Využitie kontextu sa ukázalo ako veľmi užitočné pri detekcii chodcov s nízkym rozlíšením. Samotný objekt často neposkytuje dostatočnú informáciu na jeho určenie. Ako je vidieť na obrázku 3.2, nízke rozlíšenie a rozmazanie objektu komplikuje identifikáciu objektu aj pre nás ľudí. Využitie kontextu teda môže zlepšiť výkon detekcie čo dokázali autori tejto práce. Porovnanie výkonu so štandardnou Faster R-CNN môžeme vidieť v tabuľke 3.1.

Model	**mAP**
Faster R-CNN	74.2
Faster R-CNN+LCCNET	76.3
Faster R-CNN+LCCNET+Inception-RPN	76.6

Tabuľka 3.1: Porovnanie štandardnej Faster R-CNN a rozšírení navrhnutých v práci [27] na základe dosiahnutej mAP (*mean Average Precision*).

3.3 Retinanet + RFB

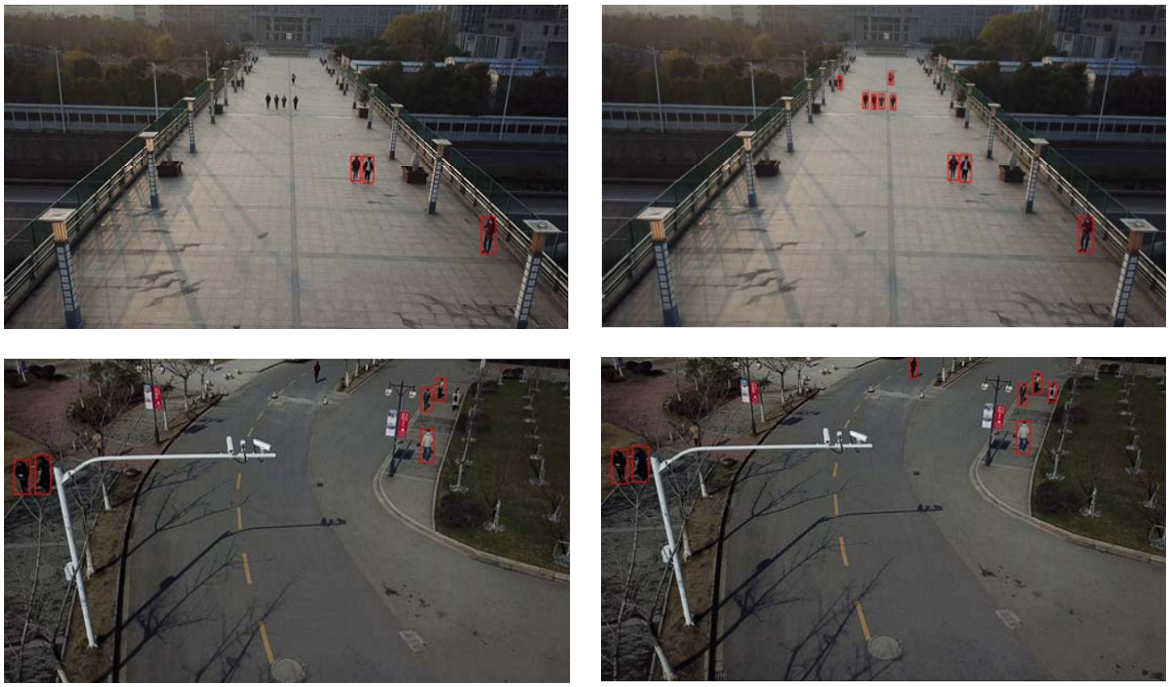
V práci [28] bol model upravený špeciálne pre detekciu chodcov na snímkoch z dronov. Autori aplikovali niekoľko úprav aby zlepšili detekciu malých objektov:

- **Úprava extraktoru príznakov - Resnet-50**. Konvolučná vrstva 7x7 bola zmenená na tri konvolučné vrstvy 3x3 nasledované aktivačnou funkciou.
- **Anchor boxes** boli zmenené z {32x32, 64x64, 128x128, 256x256, 512x512} na {8x8, 16x16, 32x32, 64x64, 128x128, 256x256}
- Použitie **RFB modul** inšpirovaného ľudským vizuálnym systémom kvôli zväčšeniu receptive field.

Model	**Base framework**	**Inception v2**	**ResNet50**	**Improved ResNet50**	**Anchors for small targets**	**RFB Module**	**Focal Loss** α **=0.75**	**AP**
Model1	SSD	√						27.38%
Model 2	Faster R-CNN							63.22%
Model 3	YOLOv3							66.62%
Model 4	RetinaNet		√					51.29%
Model 5	Our model			√				51.49%
Model 6				√	√			71.66%
Model 7				√	√	√		73.21%
Model 8				√	√	√	√	84.05%

Obr. 3.3: Porovnanie presnosti detekcie chodcov upraveného modelu Retinanet z práce [28] s inými modelmi.(prevzaté z [28])

Vďaka týmto zmenám dokázali autori zvýšiť presnosť (AP) modelu Retinanet z 51.29% na 84.05%. Tieto výsledky dosiahli na datasete vytvorenom zo 64115 obrázkov z datasetu MSCOCO2017, ktoré obsahovali chodcov. Tieto výsledky porovnali aj s inými detektormi. Výsledky porovnanie sú zobrazené na obrázku 3.3.

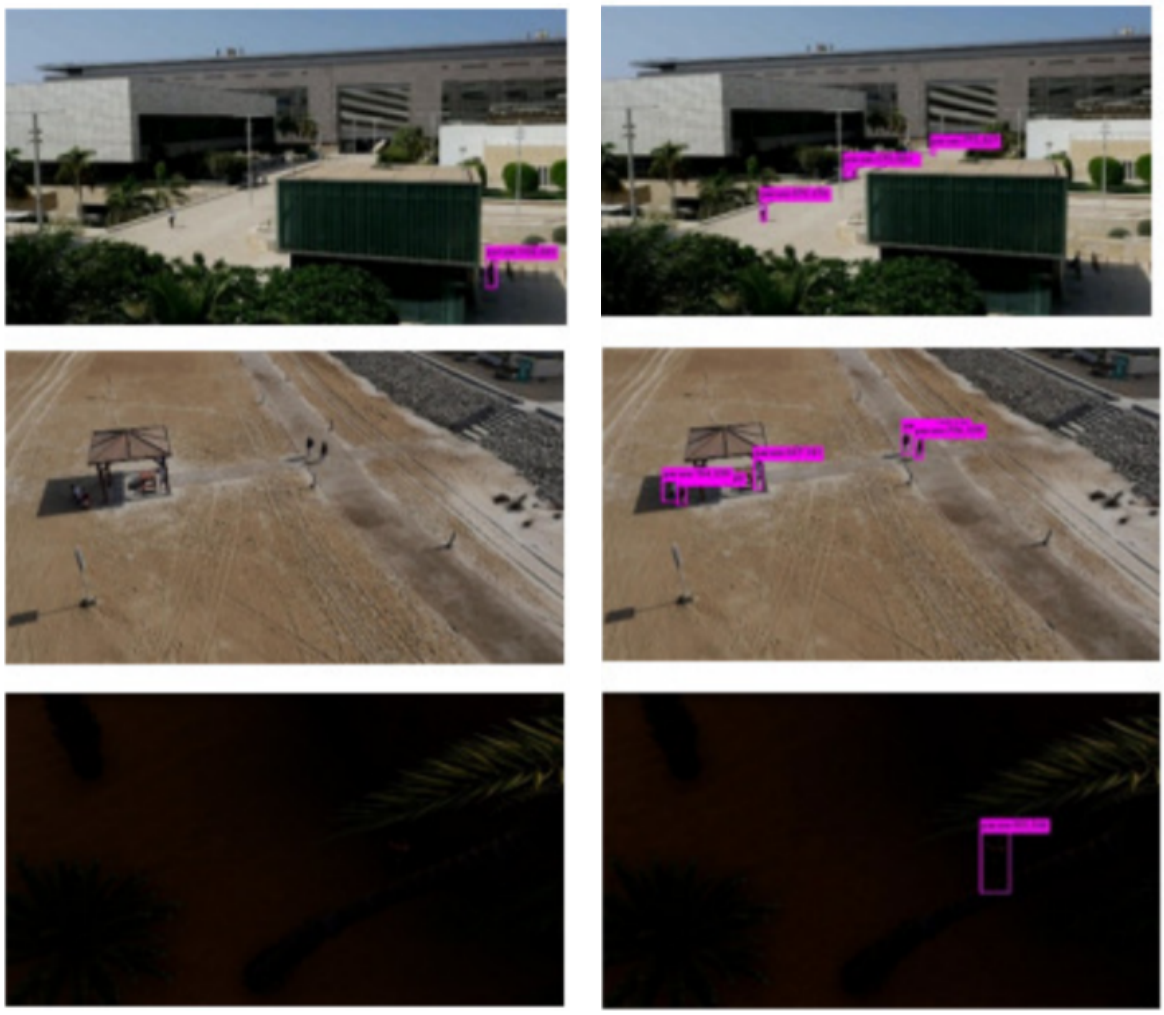


Obr. 3.4: Porovnanie detekcí pomocou Retinanet (vľavo) a upraveného modelu z práce [28]. (prevzaté z [28])

Dataset obsahoval obrázky na rôznych scénach. Porovnanie výsledkov detekcie štandardného modelu Retinanet s vylepšeným modelom sú zobrazené na obrázku 3.4. Je vidieť že vylepšený model dokázal zachytiť aj menších, ťažko rozpoznateľných chodcov.

3.4 UAV-YOLO

Autori článku [29] sa pokúsili upraviť existujúci model YOLOv3 špeciálne na detekciu malých objektov na snímkoch z dronov. Tento model nazvali **UAV-YOLO**. Úprava spočívala v zmene štruktúry siete Darknet. Resblock je optimalizovaný spojením dvoch ResNet jednotiek, ktoré majú rovnakú výšku a šírku. Ďalšou úpravou bolo zväčšenie konvolučných operácii v prvých vrstvách. Obe tieto úpravy viedli k zväčšeniu *receptive field*. Dataset bol rozdelený na tri skupiny: *"far", "normal",* a *"far"* v závislosti od vzdialenosti z akej bol snímok vytvorený. Výsledky a porovnanie detekcie na týchto skupinách môžeme vidieť na obrázku 3.6.



(a) (b)

Obr. 3.5: Výsledky detekcie (a)YOLO-v3 a (b)YOLOv3.

	UAV-Viewed		Normal		Games		Far	
Optimized Method	**mAP/%**	**IOU/%**	**mAP/%**	**IOU/%**	**mAP/%**	**IOU/%**	**mAP/%**	**IOU/%**
UAV-YOLO	90.86	80.42	90.90	84.11	90.62	76.54	64.42	68.02
YOLOv3	90.89	80.29	90.90	83.85	90.48	74.11	61.72	61.84
SSD300	89.87	72.34	90.68	76.45	89.19	68.21	56.01	52.98
SSD512	90.92	74.23	90.89	78.86	90.71	70.08	61.09	56.84

Obr. 3.6: Porovnanie YOLO-UAV s YOLOv3 a SSD.(prevzaté z [29])

Kapitola 4

Datasety

Pri procese učenia zohráva dataset kľúčovú rolu. Pre efektívne trénovanie a presné výsledky je potrebné mať správne označený trénovací aj testovací dataset. Počas trénovania sa používa aj tzv. validačný dataset s pomocou ktorého sa sieť snaží predchádzať pretrénovaniu.

Datasety si môžme vytvárať pomocou rôznych anotovacích nástrojov, ktoré dokážu generovať anotácie v najpoužívanejších formátoch. Príkladom takéhoto nástroja je open-source online softwér MakeSense [1]. Toto vytváranie datasetov môže byť časovo veľmi náročné, preto existujú mnohé úložiska datasetov obsahujúce veľké množstvo objektov a tried. Medzi najpopulárnejšie úložiská patrí COCO (*Common Objects in Context*), Pascal VOC (*The Pascal Visual Object Classes*), Google Open Images, Kaggle, ImageNet. Napriek tomu že tieto úložiská obsahujú mnoho dát zachytávajúcich chodcov, nenašiel som žiadny špecifický dataset, ktorý by obsahoval chodcov zachytených dronom pri typickom pohľade z veľkej výšky. Našiel som však iné zdroje ktoré takéto dáta obsahujú:

4.1 Stanford Drone Dataset (SDD)

SDD obsahuje celkom 60 videozáznamov vo formáte mov z 8 rôznych lokalít z kampusu Standfordskej Univerzity. Príklady týchto lokalít je možné vidieť na obrázku 4.1. Veľkosť datasetu je 72 GiB a spolu obsahuje približne 4,5 hodiny videozáznamu. Dataset vznikol ako súčasť práce [30], kde sa jej autori zaoberali predpovedaním trajektórie chodcov, cyklistov, aut a pod.

V SDD je zaznamenávaných 6 rôznych tried objektov: *chodec, cyklista, skateboardista, golfový vozík, auto autobus*. Ich percentuálne zastúpenie v jednotlivých lokalitách je zaznamenané v tabuľke 4.1.

Pre každý videozáznam existuje anotačný súbor `annotation.txt` kde sú zaznamenané polohy objektov pre každý snímok vo videozázname. Okrem čísla snímku, polohy a triedy objektu obsahujú anotácie aj 3 príznaky: *occluded*, *lost* a *generated*. Tieto príznaky sú zaznamenané kvôli účelu za akým bol dataset vytvorený - monitorovanie trajektórie objektov. Príznak *occluded* znamená že

[1] Aplikácia je dostupná na https://www.makesense.ai.

Lokalita	**Cyklista**	**Chodec**	**Skateboardista**	**Golfový vozík**	**Auto**	**Autobus**
gates	51.94	43.36	2.55	0.29	1.08	0.78
little	56.04	42.46	0.67	0	0.17	0.67
nexus	4.22	64.02	0.60	0.40	29.51	1.25
coupa	18.89	80.61	0.17	0.17	0.17	0
bookstore	32.89	63.94	1.63	0.34	0.83	0.37
deatCircle	56.30	33.13	2.33	3.10	4.71	0.42
quad	12.50	87.50	0	0	0	0
hyang	27.68	70.01	1.29	0.43	0.50	0.09

Tabuľka 4.1: Percentuálne zastúpenie tried v jednotlivých lokalitách SDD.

(a) (b)

(c) (d)

Obr. 4.1: Ukážky zo 4 lokalít SDD: (a) bookstore, (b) gates, (c) little, (d) deathCircle

objekt niečo prekrýva, napr. strom. Príznak *lost* hovorí že objekt sa nachádza mimo snímaný záber a nakoniec *generated* znamená že anotácia bola vygenerovaná automaticky. Niektoré anotácie boli vytvorené manuálne, väčšina však bola automaticky vygenerovaná. Dá sa teda predpokladať že dataset by mohol obsahovať chybné anotácie.

4.2 VisDrone

Visdrone-2019 je dataset vytvorený v rámci prác [31] a [32] a slúži ako podklad v súťaži Vision Meets Drone Challenge. Táto súťaž sa sústredí na 5 hlavných úloh:

1. Detekcia objektov v obrázkoch.
2. Detekcia objektov vo videu.
3. Sledovanie jedného objektu. Táto úloha spočíva v odhadovaní polohy objektu v nasledujúcich snímkoch.
4. Sledovanie viacerých objektu. V tejto úloha sa vyžaduje predpovedanie polohy všetkých objektov.
5. Počítanie osôb v dave.

Obr. 4.2: Ukážka z datasetu Visdrone-2019 použitého pre úlohu 1 (Detekcia objektov v obrázkoch)

Dataset pozostáva z 288 videozáznamov tvorených 261 908 snímkami a z 10 209 statických obrazov zachytených dronmi čo z neho robí pravdepodobne najrozsiahlejší dataset tohto druhu. Záznamy boli zachytené v 14 rôznych čínskych mestách, v rôznych prostrediach (mestských a prímestských) za rôzneho počasia a svetelných podmienok. Všetky anotácie boli vytvorené manuálne a ich celkový počet je približne 2.6 milióna

Dataset je podľa vyššie spomenutých úloh rozdelený na 5 častí. Každá z týchto častí je ďalej rozdelená na trénovací, testovací a validačný dataset. Pre úlohu 1 (Detekcia objektov v obrazoch) exituje 10209 obrázkov v ktorých je zaznamenávaných niekoľko tried objektov: *chodec*, *osoba*[2], *auto*, *van*, *autobus*, *kamión*, *motocykel*, *auto*, *trojkolka*, a *zastrešená trojkolka*. Anotácie obsahujú aj príznak *occlusion* ktorý môže mať hodnoty 0,1 a 2 podľa toho či je objekt prekrytý, čiastočne prekrytý alebo nie je prekrytý.

4.3 P-DESTRE

P-DESTRE dataset je výsledkom spolupráce výskumných pracovníkov z Univerzity v Beira v Portugalsku a Univerzity vedy a technológie JSS v Indii. Dataset bol vytvorený za účelom detekcie, sledovania ale aj reidentifikácie osôb. Práve kvôli reidentifikácii osôb obsahuje zábery z menšej výšky (5,5m a 6.7m) v porovnaní s SDD a VisDrone-2019, ako je vidieť na ukážke z datasetu na obrázku 4.3.

Obr. 4.3: Ukážka z datasetu P-DESTRE s vyznačenými ohraničujúcimi boxmi.(prevzaté z [33])

Chodci sú kamerovaní pod uhlami v rozmedzí od 45° do 90°. Anotácie sú kvôli reidentifikácii o niečo detailnejšie. Obsahujú okrem polohy chodcov aj atribúty ako napr. *farba vlasov*, *etnicita*,

[2]Pokiaľ človek stojí alebo kráča je označený ako chodec, v opačnom príde je označený ako osoba.

obuv a pod.. Celkový počet anotácii v datasete je približne 14 miliónov, okrem toho obsahuje 261 známych identít osôb. Dáta boli nahrávané v rozlíšení 3 840 x 2 160 a uložené vo formáte mp4. [33].

Kapitola 5

Experimenty

V tejto práci som sa rozhodol na detekciu chodcov na snímkoch z dronov použiť dve architektúry detekčný sietí - Retinanet a YOLOv5 a porovnať ich úspešnosť. Pri výbere architektúr som zohľadňoval hlavne rýchlosť a presnosť detekcie akú dosahujú. Obe tieto siete by vďaka svojej rýchlosti mali zvládnuť aj detekciu v reálnom čase. Čo sa týka presnosti, napriek tomu že sú to jednostupňové detektory, v mnohých testoch dokázali obe siete prekonať aj detektory dvojstupňové. Pri porovnávaní výkonu som bral do úvahy tri hlavné ukazatele - presnosť (mAP) a rýchlosť detekcie a čas potrebný na trénovanie.

Pri trénovaní neurónových sietí prebieha veľké množstvo jednoduchých výpočtov. Tieto výpočty na sebe nie sú závislé a môžu prebiehať paralelne. Trénovanie môže byť teda urýchlené použitím grafických kariet, ktoré obsahujú tisícky jadier kde tieto výpočty môžu prebiehať. Trénovanie prebiehalo na stolovom počítači s grafickou kartou NVIDIA GeForce RTX 2080 Ti s kapacitou 10 988 MiB. Ako operačný systém som použil Ubuntu 20.04. Efektívne využitie grafickej karty zabezpečuje technológia CUDA. ktorá bola vo verzii CUDA 10.1 nainštalovaná spolu s ovládačom grafickej karty 418.181.07. Pre účely trénovania neurónových sietí existuje aj knižnica cuDNN (CUDA Deep Neural Network library) ktorá poskytuje odladené implementácie štandardných úloh ako napr. konvolúcia, pooling a normalizácia.

5.1 Výber a úprava datasetu

Z datasetov popísaných v kapitole 4 som pre účely tejto práce vybral Stanford Drone Dataset (SDD) pretože som sa chcel zamerať hlavne na detekciu chodcov z väčšej výšky. Dataset Visdrone-2019 obsahoval mnoho snímkov cestnej premávky bez chodcov a bol by vhodný pre analýzu cestnej premávky. Dataset P-DESTRE poskytuje snímky z menšej výšky keďže bol vytvorený za účelom reidentifikácie osôb.

SDD pozostáva s videozáznamov s frekvenciou snímkov 30 fps. Pre trénovanie nebolo potrebné vyberať každý snímok z videozáznamov pretože viaceré snímky za sebou sa zmenia len málo a

neposkytujú teda žiadne nové informácie z ktorých by sa mohla sieť učiť. Vyberal som teda každý 60. snímok (cca každé 2 sekundy). Extrahovanie snímkov videa som realizoval pomocou skriptov v jazyku Python a knižnice OpenCV.

Anotácie SDD sú uložené v textovom súbore v tvare: `trackID xmin ymin xmax ymax frame lost occluded generated label`. Bolo nutné ich previesť do tvarov ktoré dokážu vybraté implementácie spracovať. Tvary anotácii pre konkrétne implementácie sú popísané v podkapitolách 5.2 a 5.3. Anotácie obsahovali rôzne triedy, preto boli vybraté len tie s triedou chodec. Zároveň bolo potrebné vynechať tie anotácie ktoré obsahovali príznak *occluded* (objekt je zakrytý) a *lost* (objekt sa nachádza mimo snímku). Na manipuláciu s anotáciami a ich ukladanie do csv súborov bola použitá knižnica Pandas v jazyku Python.

Celý dataset potom obsahoval 8 163 obrázkov. Tie som rozdelil na tri časti - trénovací (80 %), testovací (10%) a validačný dataset(10%). Celkový počet obrázkov a anotácii v jednotlivých datasetoch je uvedený v tabuľke 5.1.

Dataset	Počet obrázkov	Počet anotácii
Trénovací	6 530	51 305
Testovací	816	6336
Validačný	817	6 6556
Spolu	**8 163**	**64 197**

Tabuľka 5.1: Rozdelenie datasetu

.

5.2 Trénovanie Retinanet

U architektúry Retinanet som zvoli implementáciu od holandskej spoločnosti Fizyr, ktorá sa zoberá vývojom softwéru v oblasti počítačového videnia pre logistické prostredie. Implementácia je napísaná v jazyku Python voľne dostupná na repozitári GitHub[1]. Pre svoje fungovanie vyžaduje knižnice **Keras** a **Tensorflow**. Nainštalované boli knižnice Keras 2.4. a Tensorflow 2.3, ktorý podporuje využite grafických kariet a je kompatibilný s verzia CUDA 10.1.

Anotácie sú požadované v tvare `path/to/image.jpg, xmin, ymin, xmax, ymax, class_name`. Pôvodné anotácie boli prevedené do požadovaného tvaru a uložené do troch csv súborov `train.csv`, `train.csv` a `test.csv` pre každý dataset. Všetky obrázky boli uložené do jednej zložky s názvom `images`. Okrem anotácii bol vytvorený aj csv súbor obsahujúci názvy a indexy tried v tvare `class_name, id`.

Samotné trénovanie bolo spustené pomocou skriptu `keras_retinanet/bin/train.py`. V parametroch skriptu sme špecifikovali cestu k csv súborom anotáciami pre trénovací aj validačný dataset

[1]Implementácia je dostupná na https://github.com/fizyr/keras-retinanet

a súbor obsahujúci názvy a index tried. Ďalej sme museli určili veľkosť dávky (*batch size*), čo znamená počet obrázkov ktoré sú sieti predložené v jednom kroku. Počet krokov sa potom určí podľa nasledujúcej rovnice:

$$steps = \frac{počet\ obrázkov}{batch\ size} \tag{5.1}$$

Po skončí každej epochy bol uložený model v zložke `snapshots`. Pred použitím modelu na testovanie alebo predikciu je potrebné ho konvertovať do do príslušného formátu pomocou skriptu `keras_retinanet/bin/convert_model.py`.

Rovnako ako na trénovanie obsahuje retina aj skript `keras_retinanet/bin/evaluate.py` určený na vyhodnotenie modelu. Skript pre nás vypočíta presnosť mAP a priemerný čas detekcie (*inference time*). Ako parameter bolo nutné vložiť cestu k csv súboru obsahujúcemu anotácie pre testovací dataset. Ďalším parametrom ktorý som používal bol *score-threshold* ktorý určoval minimálne dôveryhodnostné skóre s akým je objekt priradený k danej triede.

Pre dosiahnutie optimálnej presnosti som vykonal viac trénovaní, ktoré popisujem v nasledujúcich podkapitolách.

Prvé trénovanie

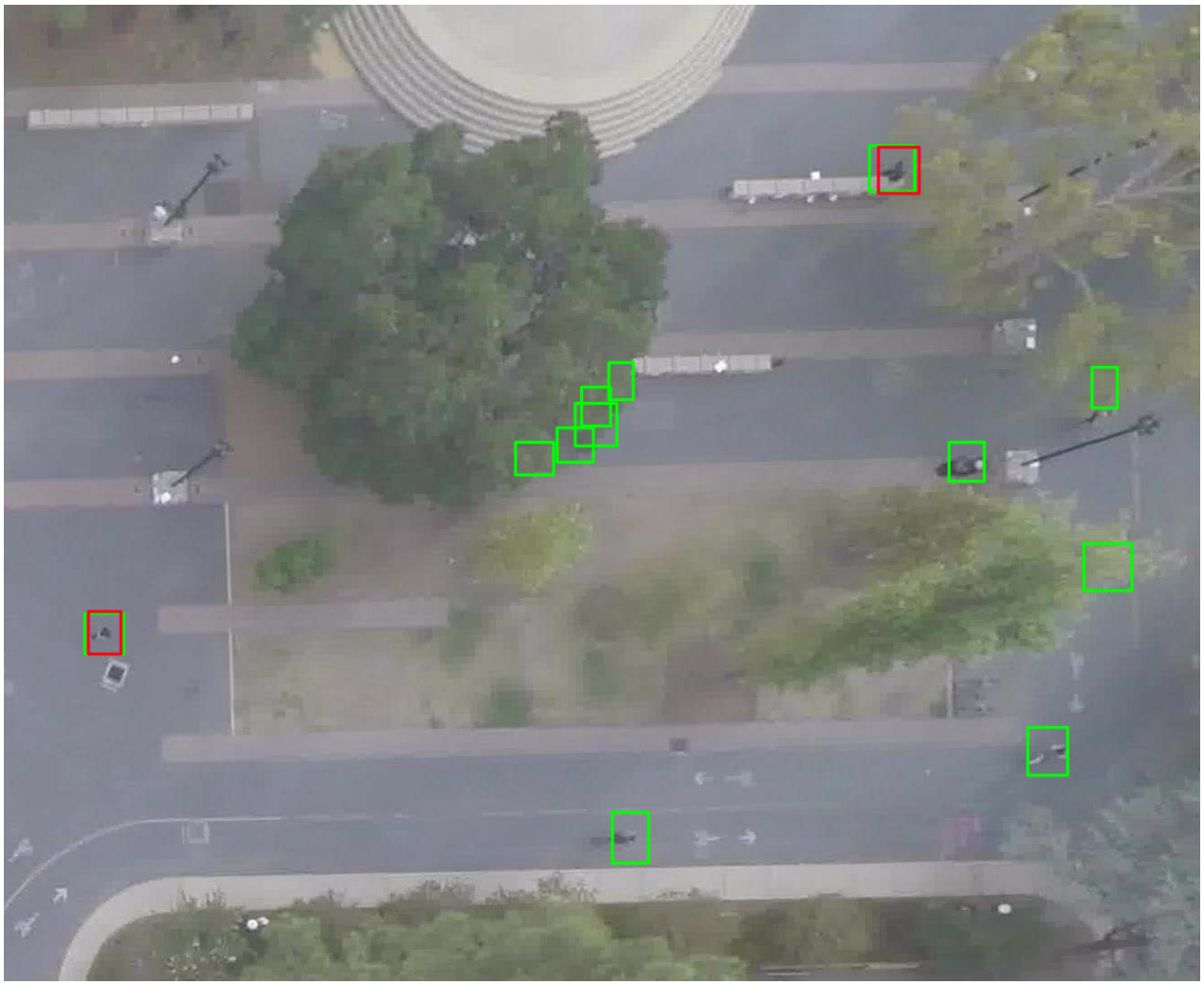
Pri prvom trénovaní som pôvodne zvolil dávku o veľkosti 4, na čo však nestačila pamäť grafickej karty. Preto som musel zmeniť veľkosť dávky na 2. Pri počte obrázkov 6 530 bol počet krokov určený na 3 265. Počet epoch, teda koľko krát bude sieti predložený celý dataset, som stanovil na 40. Ako vnútornú sieť som ponechal Resnet50 pričom sme použili predtrénovaný model na datasete COCO, ktorý je rovnako dostupný na repozitári[2]. Priebeh trénovania bolo je možné sledovať v príkazovom riadku. Trénovanie trvalo približne 10 hodín.

Model bol konvertovaný a vyhodnotený na testovacom datasete s minimálnym dôveryhodnostným skóre stanoveným na 0,5. Model dosiahol čas detekcie **71 ms** a veľmi nízku presnosť **0.28 mAP**. Všetky vykonané detekcie boli uložené znázornené na obrázkoch a uložené. Po ich dôkladnej kontrole bolo zistené že dataset obsahuje množstvo chybných anotácii. Niektoré anotované objekty sú zakryté čo môžeme vidieť na obrázku 5.1. Taktiež množstvo cyklistov bolo nesprávne anotovaných ako chodci. To mohlo byť negatívny vplyv na presnosť modelu, čo dokazuje aj fakt že niektorý cyklisti boli detekovaní ako chodci. Z uvedeného vyplýva že dataset musel byť upravený.

Druhé trénovanie

Pre druhé trénovanie bol vytvorený nový dataset. V pôvodnom datasete bol použitý každý 60. snímok z videozáznamov. Videozáznamy majú rôznu dĺžku a každý z nich poskytuje iné svetelné podmienky, lokalitu výšku z akej je snímok zachytený a podobne.Niektoré záznamy potom mohli byť sieťou preferované. Z toho dôvodu som z každého videozáznamu vybral rovnaké množstvo snímkov

[2]Predtrénované modely na datasete COCO sú dostupné na https://github.com/fizyr/keras-retinanet/releases



Obr. 5.1: Výsledok detekcie po prvom trénovaní. Anotácie sú vyznačené zelenou farbou, červenou farbou sú označené detekcie.

pre trénovací, testovací a validačný dataset. Do anotácii som zahrnul aj triedu skateboardista pod jednou triedou ako chodec pretože z danej výšky je len veľmi ťažko rozlíšiteľný od chodcov. Ďalej bolo nutné všetky obrázky skontrolovať manuálne a odstrániť chybné anotácie. Pre túto kontrolu bola vytvorená aplikácia (dodatok B) s jednoduchým uživateľským rozhraním pomocou Python knižnice Tkinter. Spolu bolo odstránených až 3 873 chybných anotácii. Počet obrázkov a anotácii v nových datasetoch je uvedený v tabuľke 5.2.

Trénovanie 40 epoch trvalo približne 7 hodín. Model bol opäť vyhodnotený na testovacom datasete. Presnosť detekcie sa zvýšila na **0,38 mAP**, rýchlosť sa zmenila len nepatrne na **73 ms**. Po preskúmaní vykonaných detekcií som zistil že niektorý cyklisti sú stále chybne detekovaný ako chodci, tentokrát ale v menšom množstve. Je to spôsobené tým že cyklista je z väčšej výšky len ťažko rozoznateľný od chodca.

Dataset	Počet obrázkov	Počet anotácii
Trénovací	4 641	37 182
Testovací	540	4 099
Validačný	539	4 064
Spolu	**8 163**	**64 197**

Tabuľka 5.2: Nový dataset po odstránení chybných anotácií

.

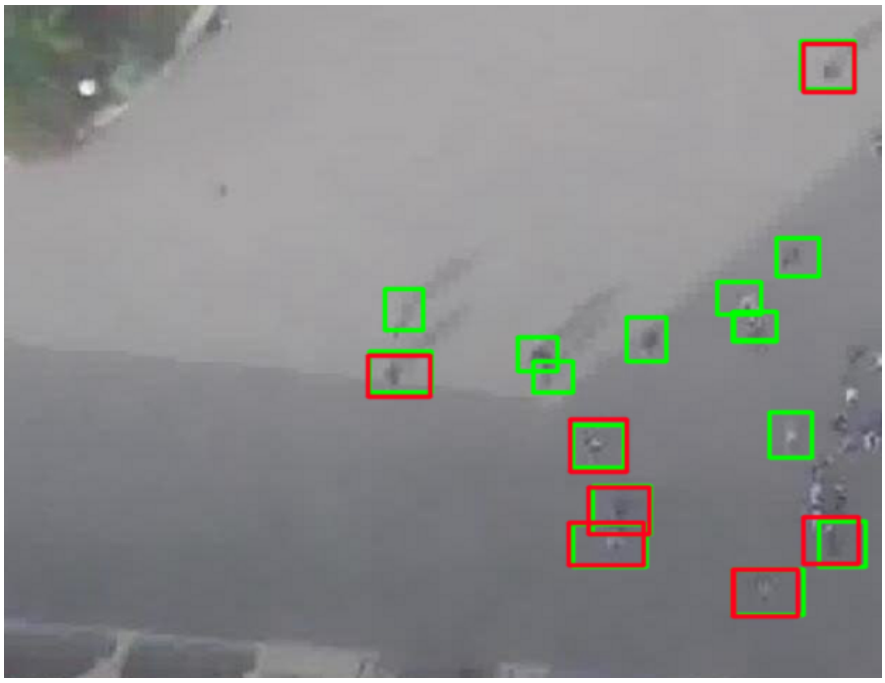
Obr. 5.2: Výsledok detekcie po druhom trénovaní. Anotácie sú vyznačené zelenou farbou, červenou farbou sú označené detekcie.

Výsledky druhého trénovania neboli uspokojivé a preto bolo vykonané ďalšie trénovanie.

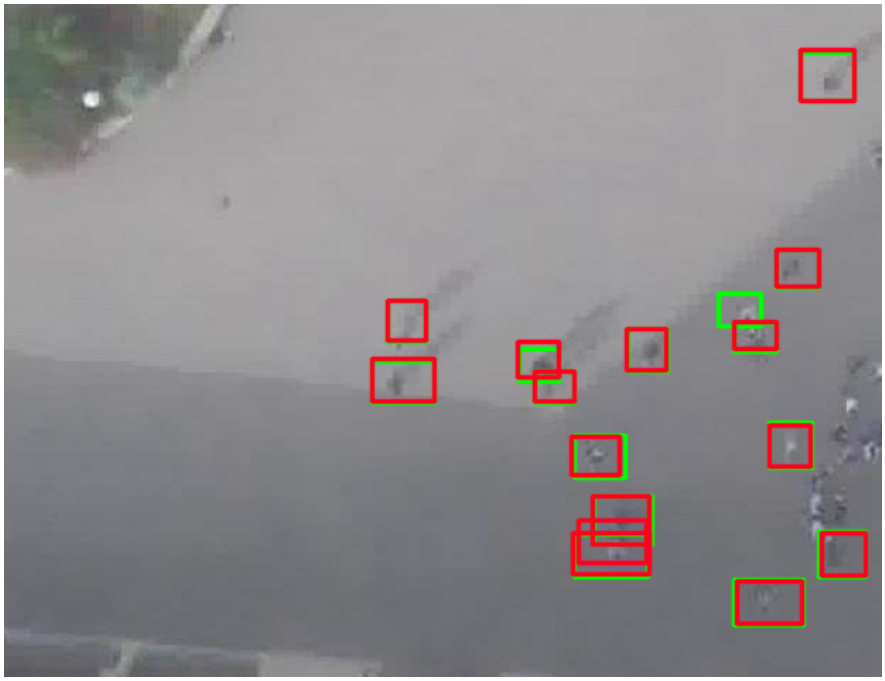
Tretie trénovanie

Pre tretie trénovanie som sa nechal inšpirovať prácou [28] a zmenil som veľkosti kotviacich boxov (anchor boxes) pre zlepšenie detekcie malých objektov. Najväčší box 512x512 bol vynechaný a pridal som box 16x16. Nové veľkosti som uložil do súboru `config.ini` ktorý bol použitý ako parameter pre trénovací skript.

Úprava kotviacich boxov sa ukázala ako užitočná. Po otestovaní natrénovaného modelu sme získali presnosť **0,49 mAP** a rýchlosť detekcie **71 ms**. Trénovanie trvalo rovnako ako predchádzajúcom prípade približne 7 hodín. Ako môžme vidieť na obrázku 5.3, druhý model nedokázal



(a)



(b)

Obr. 5.3: Porovnanie detekcie pomocou modelov Retinanet s pôvodnými kotviacimi boxmi (a) a s upravenými kotviacimi boxmi (b). Červenou farbou sú ohraničené detekcie, zelenou anotácie (*ground truth*).

narozdiel od tretieho detekovať niektoré menšie objekty. Napriek zvýšeniu presnosti v porovnaní s druhým modelom je presnosť 0,49 mAP pomerne nízka čo však môže byť spôsobené ľudským faktorom (chybné anotácie, ťažko rozpoznateľné objekty z výšky).

Model	Veľkosť obrázku (px)	mAP	Rýchlosť detekcie (ms)	Veľkosť dávky	Trvanie trénovania (h)
Retinanet		0,38	72,6	2	7,1
Retinanet + anchor boxes		0,49	70,9	2	7,1

Tabuľka 5.3: Porovnanie modelov Retinanet

.

5.3 Trénovanie YOLOv5

U YOLOv5 som zvolil zatiaľ jedinú dostupnú implementáciu od spoločnosti Ultralytics [3]. Je taktiež napísaná v jazyku Python, ale na rozdiel od frameworku Tensorflow v implementácii v podkapitole 5.2 tu bol použitý framework **PyTorch**, primárne vyvíjaný spoločnosťou Facebook. Pytorch taktiež podporuje akceleráciu výpočtov pomocou grafických kariet, preto boli nainštalované verzie Torch 1.7.1+cu101 a Torchvision 0.8.2+cu101 kompatibilné s verziou CUDA 10.1.

Pri trénovaní modelu bol použitý upravený dataset popísaný v tabuľke 5.2. YOLOv5 akceptuje anotácie v tvare `class_name xcenter ycenter width height`. V tomto prípade je ohraničujúci box definovaný súradnicami svojho stredu (`xcenter` a `ycenter`), výškou (`height`) a šírkou (`width`). Tieto hodnoty musia byť normalizované aby nadobúdali hodnoty v intervale $(0, 1)$. Normalizácia bola vykonaná pomocou kódu vo výpise 5.1.

```
def normalize_coordinates(image, xmin, ymin, width, height):
    '''
    param image: reprezentácia obrázka vo formáte nparray
    param xmin: x-súradnica ľavé horného horu ohraničujúceho boxu
    param ymin: y-súradnica ľavé horného horu ohraničujúceho boxu
    param width: šírka ohraničujúceho boxu
    param height: výška ohraničujúceho boxu
    '''

    # získame výšku a šírku obrázka
```

[3] Implementácia je dostupná na https://github.com/ultralytics/yolov5.

```
    h_img, w_img, c = image.shape

    # získame súrdanice stredu ohraničujúceho boxu a videlíme ich šírkou a výškou
        obrázka
    xcenter = (xmin + width/2) / w_img
    ycenter = (ymin + height/2) / h_img

    # vydelíme šírku a výšku boxu šírkou a výškou obrázka
    width = width / w_img
    height = height / h_img

    return xcenter, ycenter, width, height
```

Listing 5.1: Funkcia použitá na normalizáciu anotaácií pre formát YOLOv5

Pred trénovaním bol vytvorený konfiguračný súbor vo formáte `yaml`, ktorý obsahoval počet tried a cesty k anotačným súborom a obrázkom pre každý typ datasetu. V prípade potreby je tu možné špecifikovať aj iné parametre trénovania, napr. veľkosti kotviacich boxov. Pre trénovanie som zvolil 2 varianty **YOLOv5s** a **YOLOv5m** kvôli ich nižším výpočetným nárokom a vyššej rýchlosti detekcie. Každá varianta bola trénovaná dva krát s použitím dvoch veľkostí obrázku 1024x1024 pixelov a 2016x2016 pixelov a následne boli tieto výsledky porovnané. Počet epoch u každého trénovania bol zvolený na 50.

Trénovanie 2016x2016

Pre trénovanie oboch variánt s veľkosťou obrázku 2016x2016 bolo nutné zvoliť veľkosť dávky 2, pretože pre vyššiu dávku nebola kapacita grafickej karty dostatočná. Pri tejto veľkosti dávky bolo pri každej epoche trénovania varianty YOLOv5m využitých približne 9,6 GiB čo je približne 90% jej celkovej kapacity. U varianty YOLOv5s bolo využitých približne 7,2 GiB. Trénovanie varianty `m` trvalo približne 10 hodín, zatiaľ čo varianta `s` zvládla rovnaký počet epoch približne za polovicu, napriek tomu bola jej presnosť nepatrne vyššia ako u varianty `m`.

Trénovanie 1024x1024

Pri veľkosti obrázku 1024x1024 boli výpočetné nároky nižšie a bolo možné použiť veľkosť dávky 4 čo výrazne urýchlilo proces trénovania ako môžme vidieť v tabuľke 5.4. U varianty YOLOv5s trval celý proces len 1,8 hodiny. Z výsledkov v tabuľke 5.4 je evidentné že veľkosť obrázku výrazne zníži dĺžku trénovania aj čas potrebný na detekciu. Avšak presnosť sa znížila len nepatrne, dokonca v prípade Yolov5m bola dosiahnutá vyššia presnosť pri trénovaní 1024x1024 a rýchlosť sa zvýšila

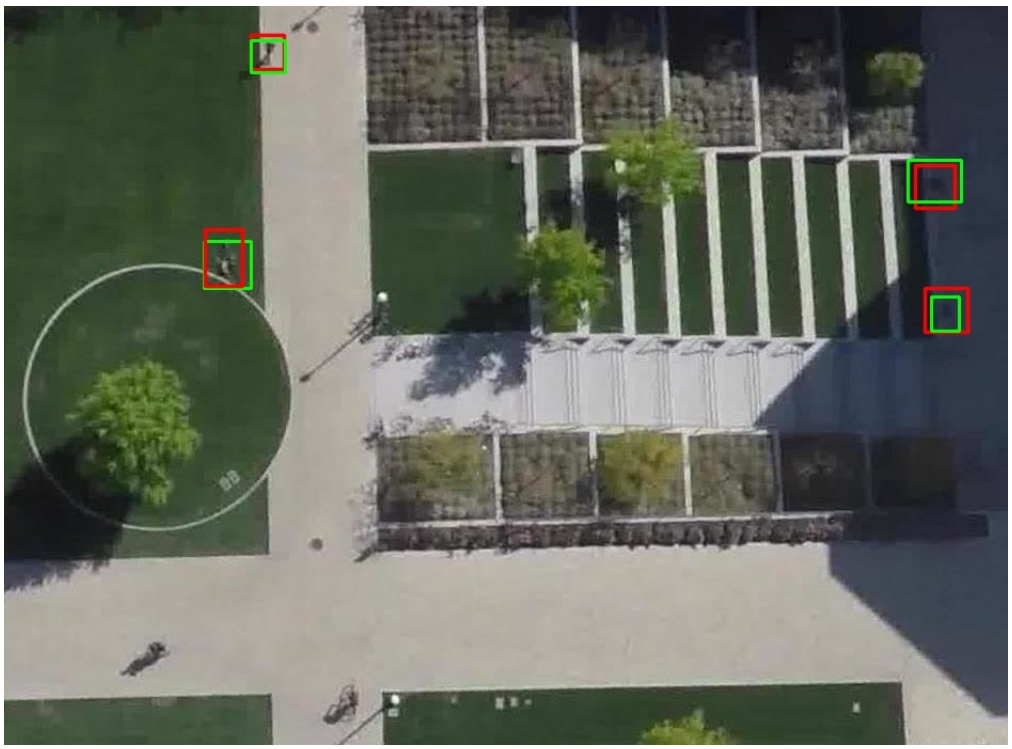
až 3-násobne. U varianty YOLOv5s bola pri trénovaní s veľkosťou 1024x1024 dosiahnutá najvyššia rýchlosť 5,6 ms.

Model	Veľkosť obrázku (px)	mAP	Rýchlosť detekcie (ms)	Veľkosť dávky	Trvanie trénovania (h)
Yolov5m	2016x2016	0.69	24,9	2	10,3
Yolov5m	**1024x1024**	**0,70**	**8,0**	**4**	**2.6**
Yolov5s	2016x2016	0,69	11,2	2	5,8
Yolov5s	1024x1024	0.68	5,6	4	1,8

Tabuľka 5.4: Porovnanie modelov YOLO

.

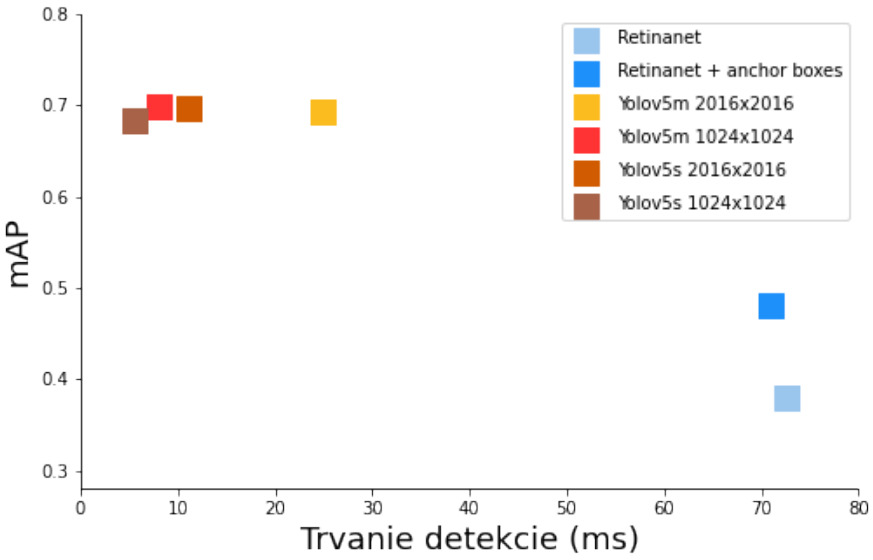
Najpresnejší bol teda model YOLOv5m s veľkosťou obrázku 1024x1024. Výsledok detekcie s použitím tohoto modelu môžme vidieť na obrázku 5.4. Na pravej strane obrázku si môžme všimnúť že model si dokázal poradiť aj s niektorými ťažko viditeľnými chodcami. Ďalej si môžme všimnúť že dokázal správne odlíšiť chodcov od cyklistov.



Obr. 5.4: Výsledok detekcie s použitím najpresnejšieho modelu - YOLOv5m 1024x1024

5.4 YOLOv5 vs Retinanet

V porovnaní s modelmi Retinanet dosiahol model Yolov5m výrazne lešpie výsledky aj napriek použitiu upravených kotviacich boxov pri treťom trénovaní Retinanet. Všetky modely YOLOv5 dosiahli oveľa vyššiu presnosť aj rýchlosť detekcie ako modely Retinanet. Okrem toho mali modely 1024x1024 výrazne nižšie výpočetné nároky a teda aj dĺžka trénovania bola výrazne nižšia. V tabuľke 5.5 a na obrázku 5.5 môžme vidieť porovnanie všetkých trénovaných modelov v tejto práci. Do týchto porovnaní neboli zahrnuté výsledky prvého trénovania Retinanet pretože boli trénované aj testované na inom datasete.



Obr. 5.5: Porovnanie modelov YOLO a Retinanet

Na obrázku 5.6 je možné sledovať porovnanie detekcií YOLOv5m 1024x1024 a modelu Retinanet s upravenými kotviacimi boxmi. Aj keď niektorých chodcov nebol schopný presne označiť ani model YOLO je vidieť že v porovnaní s modelom Retinanet dokázal zachytiť aj ťažko viditeľného chodca v tieni na pravej strane obrázku. Model YOLO dokázal taktiež na lepšie odlíšiť cyklistov od chodcov s čím mal aj najpresnejší model Retinanet na niektorých obrázkoch ťažkosti. Môžeme teda skonštatovať že pri detekcii chodcov z pohľadu výšky bol rozhodne úspešnejší model YOLOv5, čo sa týka presnosti a rýchlosti detekcie ako aj výpočetných nárokov a dĺžky trénovania.

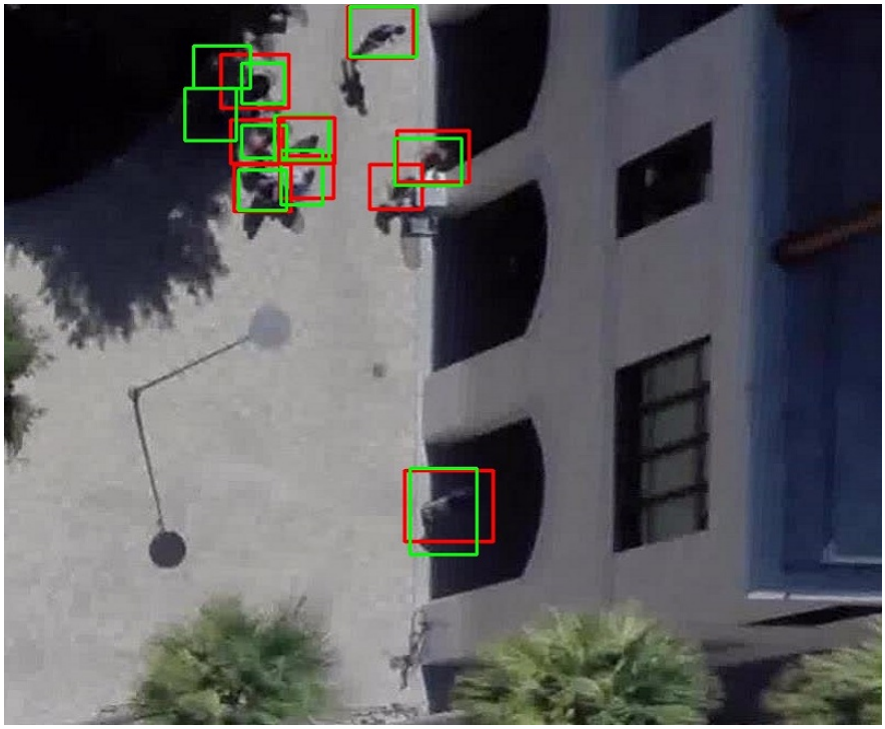
Implementácia YOLOv5 bola predstavená v roku 2020 a je teda v porovnaní s implementáciou Retinanet (2017) novšia a má teda za sebou aj dlhší vývoj. Pri trénovaní poskytuje viac možností ako napr. výber varianty podľa požadovanej rýchlosti a presnosti. Natrénované modely majú niekoľkonásobne nižšiu veľkosť (14 - 41 MiB) ako modely Retinanet (142 MiB). Veľkou výhodou

implementácie YOLOv5 je aj automatická analýza kotviacich boxov na základe anotácii, čo môže byť jeden z hlavných faktorov ktorý vplýval na presnosť modelu.

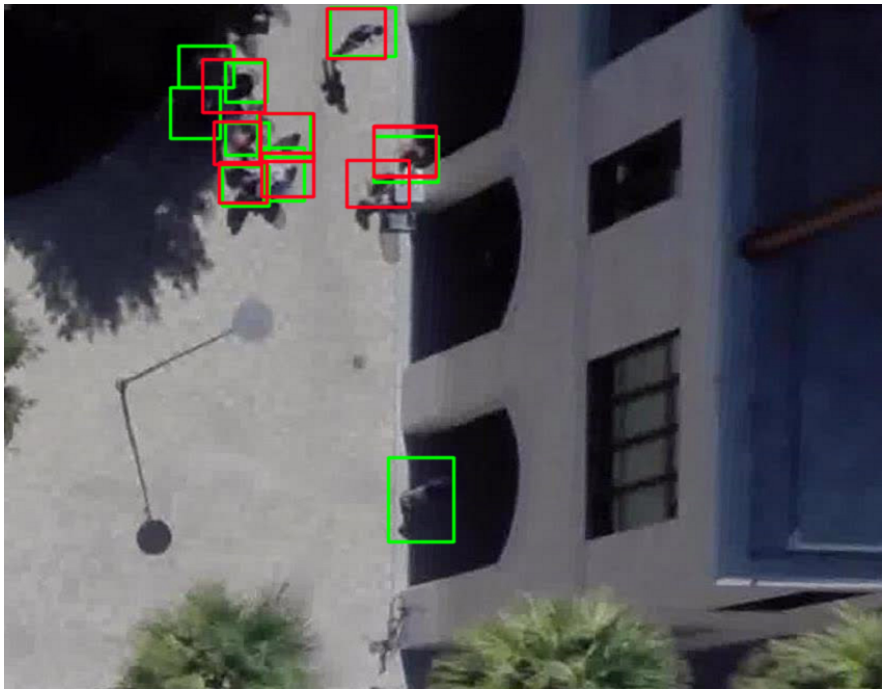
Model	Veľkosť obrázku (px)	mAP	Rýchlosť detekcie (ms)	Veľkosť dávky	Trvanie trénovania (h)
Retinanet		0,38	72,6	2	7,1
Retinanet + anchor boxes		0,49	70,9	2	7,1
Yolov5m	2016x2016	0.69	24,9	2	10,3
Yolov5m	**1024x1024**	**0,70**	**8,0**	4	**2.6**
Yolov5s	2016x2016	0,69	11,2	2	5,8
Yolov5s	1024x1024	0.68	5,6	4	1,8

Tabuľka 5.5: Porovnanie modelov YOLO a Retinanet

.



(a)



(b)

Obr. 5.6: .Porovnanie najpresnješieho modelu YOLOv5 (a) s najpresnejším modelom Retinanet (b). Červenou farbou sú ohraničené detekcie, zelenou anotácie (*ground truth*).

Kapitola 6

Záver

Cieľom mojej práce bolo porovnať a nájsť vhodnú detekčnú sieť ktorá by zvládla detekciu chodcov na snímkoch z dronov s optimálnou rýchlosťou a presnosťou. Porovnávané boli dve detekčné siete YOLOv5 a Retinanet. U Retinanet bol preukázaný vplyv kotviacich boxov (*anchor boxes*) na presnosť detekcie. Napriek zvýšeniu presnosti s ich použitím sa detektor YOLOv5 ukázal vo všetkých ohľadoch ako vhodnejší pre túto úlohu. . Ďalej bol preukázaný vplyv vhodne vytvoreného datasetu a presnosť anotácii na presnosť detekcie. Po úprave datasetu, odstránení chybných anotácii a úprave kotviacich boxov sa presnosť detektoru Retinanet zvýšila z 0.28 mAP na 0,49 mAP. Táto presnosť je stále pomerne nízka. To ale môže byť stále spôsobené niektorými chybnými alebo spornými anotáciami. Ako najpresnejší sa ukázal model YOLOv5m s veľkosťou obrázku 1024x1024 s presnosťou až 0,70 mAP. Čo sa týka rýchlosti, bol najúspešnejší model YOLOv5s ktorý zvládol detekciu iba za 5.6 ms.

Literatura

1. BERNAMA. *Over 1,800 teenagers, children went missing last year* [online]. 2017-11 [cit. 2021-04-14]. Dostupné z: https://www.nst.com.my/news/nation/2017/11/303741/over-1800-teenagers-children-went-missing-last-year.
2. *Deep Learning Drone Detects Fights, Bombs, Shootings in Crowds* [online]. 2018-07 [cit. 2021-04-19]. Dostupné z: https://thenewstack.io/deep-learning-drone-detects-fights-bombs-shootings-in-crowds/.
3. LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998-11, roč. 86, č. 11, s. 2278–2324. ISSN 1558-2256. Dostupné z DOI: 10.1109/5.726791. Conference Name: Proceedings of the IEEE.
4. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]* [online]. 2015-12 [cit. 2021-04-03]. Dostupné z: http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.
5. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* [online]. 2017-05, roč. 60, č. 6, s. 84–90 [cit. 2021-03-19]. ISSN 0001-0782, ISSN 1557-7317. Dostupné z DOI: 10.1145/3065386.
6. SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]* [online]. 2015-04 [cit. 2021-03-21]. Dostupné z: http://arxiv.org/abs/1409.1556. arXiv: 1409.1556.
7. RESEARCHER PhD, Matthew Stewart. *Simple Introduction to Convolutional Neural Networks* [online]. 2020-07 [cit. 2021-03-18]. Dostupné z: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac.
8. SKALSKI, Piotr. *Gentle Dive into Math Behind Convolutional Neural Networks* [online]. 2019-04 [cit. 2021-03-19]. Dostupné z: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9.

9. ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017-08, s. 1–6. Dostupné z DOI: 10.1109/ICEngTechnol.2017.8308186.

10. YAMASHITA, Rikiya; NISHIO, Mizuho; DO, Richard Kinh Gian; TOGASHI, Kaori. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging* [online]. 2018-08, roč. 9, č. 4, s. 611–629 [cit. 2021-03-18]. ISSN 1869-4101. Dostupné z DOI: 10.1007/s13244-018-0639-9. Number: 4 Publisher: SpringerOpen.

11. SOVIANY, P.; IONESCU, R. T. Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction. In: *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 2018-09, s. 209–214. Dostupné z DOI: 10.1109/SYNASC.2018.00041.

12. UIJLINGS, J. R. R.; SANDE, K. E. A. van de; GEVERS, T.; SMEULDERS, A. W. M. Selective Search for Object Recognition. *International Journal of Computer Vision* [online]. 2013-09, roč. 104, č. 2, s. 154–171 [cit. 2021-03-21]. ISSN 0920-5691, ISSN 1573-1405. Dostupné z DOI: 10.1007/s11263-013-0620-5.

13. GANDHI, Rohith. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms* [online]. 2018-07 [cit. 2021-03-21]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

14. GIRSHICK, Ross. Fast R-CNN. *arXiv:1504.08083 [cs]* [online]. 2015-09 [cit. 2021-03-21]. Dostupné z: http://arxiv.org/abs/1504.08083. arXiv: 1504.08083.

15. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross; SUN, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]* [online]. 2016-01 [cit. 2021-03-22]. Dostupné z: http://arxiv.org/abs/1506.01497. arXiv: 1506.01497.

16. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. Mask R-CNN. *arXiv:1703.06870 [cs]* [online]. 2018-01 [cit. 2021-03-22]. Dostupné z: http://arxiv.org/abs/1703.06870. arXiv: 1703.06870.

17. LIN, Tsung-Yi; GOYAL, Priya; GIRSHICK, Ross; HE, Kaiming; DOLLÁR, Piotr. Focal Loss for Dense Object Detection. *arXiv:1708.02002 [cs]* [online]. 2018-02 [cit. 2021-04-03]. Dostupné z: http://arxiv.org/abs/1708.02002. arXiv: 1708.02002.

18. BOER, Pieter-Tjerk de; KROESE, Dirk P.; MANNOR, Shie; RUBINSTEIN, Reuven Y. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research* [online]. 2005-02, roč. 134, č. 1, s. 19–67 [cit. 2021-04-03]. ISSN 0254-5330, ISSN 1572-9338. Dostupné z DOI: 10.1007/s10479-005-5724-z.

19. REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]* [online]. 2016-05 [cit. 2021-04-04]. Dostupné z: http://arxiv.org/abs/1506.02640. arXiv: 1506.02640.

20. REDMON, Joseph; FARHADI, Ali. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]* [online]. 2016-12 [cit. 2021-04-04]. Dostupné z: http://arxiv.org/abs/1612.08242. arXiv: 1612.08242.

21. REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs]* [online]. 2018-04 [cit. 2021-04-04]. Dostupné z: http://arxiv.org/abs/1804.02767. arXiv: 1804.02767.

22. SYNCED. *YOLO Creator Joseph Redmon Stopped CV Research Due to Ethical Concerns | Synced* [online]. 2020-02 [cit. 2021-04-05]. Dostupné z: https://syncedreview.com/2020/02/24/yolo-creator-says-he-stopped-cv-research-due-to-ethical-concerns/.

23. BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]* [online]. 2020-04 [cit. 2021-04-05]. Dostupné z: http://arxiv.org/abs/2004.10934. arXiv: 2004.10934.

24. NELSON, Joseph; JUN 10, Jacob Solawetz; READ, 2020 4 Min. *YOLOv5 is Here* [online]. 2020-06 [cit. 2021-04-05]. Dostupné z: https://blog.roboflow.com/yolov5-is-here/.

25. *ultralytics/yolov5* [online]. Ultralytics, 2021-04 [cit. 2021-04-05]. Dostupné z: https://github.com/ultralytics/yolov5. original-date: 2020-05-18T03:45:11Z.

26. AGUILAR, W. G.; LUNA, M. A.; MOYA, J. F.; ABAD, V.; PARRA, H.; RUIZ, H. Pedestrian Detection for UAVs Using Cascade Classifiers with Meanshift. In: *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. 2017-01, s. 509–514. Dostupné z DOI: 10.1109/ICSC.2017.83.

27. LIU, T.; FU, H. Y.; WEN, Q.; ZHANG, D. K.; LI, L. F. Extended faster R-CNN for long distance human detection: Finding pedestrians in UAV images. In: *2018 IEEE International Conference on Consumer Electronics (ICCE)*. 2018-01, s. 1–2. Dostupné z DOI: 10.1109/ICCE.2018.8326306. ISSN: 2158-4001.

28. XIANG, C.; SHI, H.; LI, N.; DING, M.; ZHOU, H. Pedestrian Detection Under Unmanned Aerial Vehicle an Improved Single-Stage Detector Based on RetinaNet. In: *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 2019-10, s. 1–6. Dostupné z DOI: 10.1109/CISP-BMEI48845.2019.8965666.

29. LIU, Mingjie; WANG, Xianhao; ZHOU, Anjian; FU, Xiuyuan; MA, Yiwei; PIAO, Changhao. UAV-YOLO: Small Object Detection on Unmanned Aerial Vehicle Perspective. *Sensors* [online]. 2020-01, roč. 20, č. 8, s. 2238 [cit. 2021-04-05]. Dostupné z DOI: 10.3390/s20082238. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

30. ROBICQUET, Alexandre; SADEGHIAN, Amir; ALAHI, Alexandre; SAVARESE, Silvio. Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes. In: LEIBE, Bastian; MATAS, Jiri; SEBE, Nicu; WELLING, Max (ed.). *Computer Vision – ECCV 2016* [online]. Cham: Springer International Publishing, 2016, zv. 9912, s. 549–565 [cit. 2021-04-07]. ISBN 978-3-319-46483-1 978-3-319-46484-8. Dostupné z DOI: 10.1007/978-3-319-46484-8_33. Series Title: Lecture Notes in Computer Science.

31. ZHU, Pengfei; WEN, Longyin; DU, Dawei; BIAN, Xiao; HU, Qinghua; LING, Haibin. Vision Meets Drones: Past, Present and Future. *arXiv:2001.06303 [cs]* [online]. 2020-07 [cit. 2021-04-07]. Dostupné z: http://arxiv.org/abs/2001.06303. arXiv: 2001.06303.

32. ZHU, Pengfei; WEN, Longyin; BIAN, Xiao; LING, Haibin; HU, Qinghua. Vision Meets Drones: A Challenge. *arXiv:1804.07437 [cs]* [online]. 2018-04 [cit. 2021-04-07]. Dostupné z: http://arxiv.org/abs/1804.07437. arXiv: 1804.07437.

33. KUMAR, S. V. Aruna; YAGHOUBI, Ehsan; DAS, Abhijit; HARISH, B. S.; PROENÇA, Hugo. The P-DESTRE: A Fully Annotated Dataset for Pedestrian Detection, Tracking, Re-Identification and Search from Aerial Devices. *arXiv:2004.02782 [cs]* [online]. 2020-04 [cit. 2021-04-07]. Dostupné z: http://arxiv.org/abs/2004.02782. arXiv: 2004.02782.

Dodatok A

Odovzdaná príloha - detection.zip

Táto príloha obsahuje potrebné súbory k otestovaniu detekcie, ako aj k trénovaniu a vyhodnoteniu modelu. Obsah prílohy:

- **`yolov5`** - mierne upravená implemenácia yolov5. V zložke `yolov5/data` sa náchadza mnou vytvorený súbor `pedestrian.yaml` ktorý obsahuje konfiguráciu pre trénovanie.
- **`predict.py`** - zdrojový kód ktorý spustí detekciu.
- **`guide.txt`** - návod k použitiu. Obsahuje aj zoznam potrebných python knižníc.
- **`sample_images`** - niekoľko obrázkov z testovacieho datasetu. Celý dataset je dostupný na https://drive.google.com/file/d/1MFytdPdzqsTAOz4rUSmlSwyR1W3J2Wst/view?usp=sharing.
- **`weights`** - v tejto zložke sa nachádza najlepší vytrénovaný model `best.pt`.
- **`weights`** - do tejto zložky sa budú ukladať výsledky detekcie

Dodatok B

Odovzdaná príloha - fix_annots.zip

V tejto prílohe sa nachádza spustiteľný zdrojový kód k aplikácii spomenutej v kapitole 5.2, pre odstraňovanie chybných detekcií z datasetu. Obsah prílohy:

- **`main.py`** - súbor ktorý spustí aplikáciu.
- **`utils.py`** - súbor ktorý obsahuje niektoré funkcie pre prácu s anotáciami.
- **`guide.txt`** - návod na spustenie a ovládanie aplikácie. Obsahuje aj zoznam potrebných python knižníc.
- **`sample`** - v tejto zložke sa nachádza niekoľko anotácií a obrázkov pre otestovanie aplikácie. Celý dataset je dostupný na odkaze https://drive.google.com/file/d/1yY23gWG280LPEubzuL7n25Q9h0UgAkKh/view?usp=sharing.