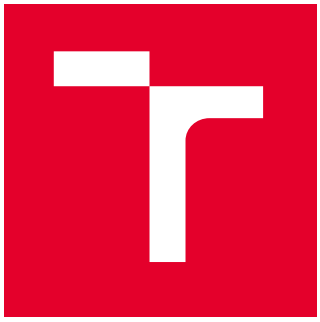


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

IDENTIFIKACE ÚČASTNÍKŮ SILNIČNÍHO PROVOZU POMOCÍ METOD HLUBOKÉHO UČENÍ

IDENTIFICATION OF ROAD USERS USING DEEP LEARNING METHODS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Marek Provazník**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jakub Kůdela, Ph.D.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Marek Provazník**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Identifikace účastníků silničního provozu pomocí metod hlubokého učení

Stručná charakteristika problematiky úkolu:

Hluboké učení a použití hlubokých neuronových sítí je jedním z aktuálně nejpoužívanějších přístupů pro detekci objektů.
Student se seznámí s problematikou aplikace těchto metod na identifikaci účastníků silničního provozu. Součástí práce pak bude rozbor používaných metodologií a implementace vhodného přístupu.

Cíle diplomové práce:

Rozbor úlohy identifikace účastníků silničního provozu.
Rešerše používaných přístupů a algoritmů z oblasti hlubokého učení.
Identifikace a rozbor vhodné datové sady.
Výběr vhodného optimalizačního algoritmu.
Implementace, verifikace a zhodnocení vybrané metody.

Seznam doporučené literatury:

REDMON, J., FARHADI, A. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

LECUN, Y., BENGIO, Y., HINTON, G. Deep learning. Nature 521.7553, pp. 436-444, 2015.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A. Deep learning. MIT press, 2016.

ABSTRAKT

Tato práce se věnuje rešerši algoritmů hlubokých konvolučních neuronových sítí pro detekci objektů a jejich učení z datasetů. Po obecném přehledu se blíže zabývá algoritmem Yolo. V praktické části jsou rozebrány použité technologie, implementace tohoto algoritmu, použité datové sady a učící strategie.

ABSTRACT

This work is devoted to research of deep convolutional neural neural networks for object detection and datasets. It is focused on Yolo algorithm. The practival part discusses used technologies, implementation of Yolo algorithm, datasets and learning strategies.

KLÍČOVÁ SLOVA

počítačové vidění, hluboké učení, detekce objektů, konvoluční neuronové sítě, Yolo v.3

KEYWORDS

computer vision, deep learning, object detection, convolutional neural networks, Yolo v.3

ÚSTAV AUTOMATIZACE A INFORMATIKY

2022

BIBLIOGRAFICKÁ CITACE

PROVAZNÍK, Marek. *Identifikace účastníků silničního provozu pomocí metod hlubokého učení*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 79 s. Diplomová práce. Vedoucí práce: Ing. Jakub Kůdela, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením Ing. Jakuba Kůdely, Ph.D. a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 20. 5. 2022 .....

Marek Provazník

PODĚKOVÁNÍ

Tímto bych rád poděkoval svému vedoucímu práce Ing. Jakubovi Kůdelovi, Ph.D. za odborné vedení, skvělý přístup, vstřícnost, ochotu a cenné rady. Dále bych chtěl poděkovat své rodině a přátelům za podporu při studiu i psaní této práce.

OBSAH

1 ÚVOD

Počítačové vidění je proces získávání informací z fotek nebo videí, kterých je v současné době velmi mnoho volně přístupných na internetu. Se stále zdokonalujícími se technologiemi jsou zařízení levnější a dostupnější než kdy dřív. Téměř každý mobilní telefon má v současnosti poměrně kvalitní fotoaparát. Pořizování digitálního obrazu je nedestruktivní, jednoduchý a levný proces. Počítačové vidění je obor se širokým zaměřením a velkou perspektivou.

V současnosti je vývoj systémů počítačového vidění v pozoruhodném stádiu. Koncept jako takový existuje už od šedesátých let 20. století, ale až s příchodem výkonné výpočetní techniky poslední dekády se vyvinul do použitelné podoby. Téměř každý den se setkáváme s aplikacemi počítačového vidění a do budoucna se dá očekávat stále frekventovanější nasazování takových systémů do praxe [9].

Jedna ze zajímavých aplikací je například rekonstrukce 3D scény ze série fotek jdoucích bezprostředně za sebou. V [39] byl v roce 2006 představen systém, který díky hledání podobností v obraze vytvořil virtuální 3D model Baziliky svatého Petra pouze z fotek stažených z internetu. Detekce objektů je důležitou součástí počítačového vidění, má široké pole uplatnění od výrobního průmyslu, autonomních vozidel a medicíny až po zpracování osobních dat uživatelů, spionáž nebo aplikace ve vojenských systémech [9].

V dnešní době je doprava lidí a zboží klíčová pro udržení chodu ekonomiky a společnosti jako takové. Automobilní doprava je páteří národního hospodářství. Zajišťuje osobní přepravdu, dopravu artiklů do obchodů, výrobních podniků a dalších. S rostoucím počtem automobilů jsou stále častější dopravní zácpy hlavně ve frekventovaných hodinách ve velkých městech. Systém vytvořený lidmi pro udržení řádu na silnicích přestává být efektivní, i při dodržení všech pravidel silničního provozu je maximální průtok aut křižovatkami omezen. Autonomní vozidla by díky vzájemné komunikaci mohla tento problém vyřešit, je však nezbytné nejprve zajistit bezpečnost všech účastníků silničního provozu. Systém identifikace těchto účastníků poskytne autonomním vozidlům informaci o jejich poloze a s tím i možnost adekvátně reagovat.

V této práci je zpracována rešerše různých způsobů pro detekci objektů a příslušenství. Je představen systém detekce potenciálních účastníků silničního provozu, který je založený na neuronových sítích a hlubokém učení.

2 Počítačové vidění

Počítačové vidění je interdisciplinární vědní obor, který se zabývá vývojem metod a technik sloužících k extrakci konkrétních informací z obsahu digitálních fotografií a videí. Problémy počítačového vidění jsou mnohdy snadno řešitelné až triviální pro lidi, dokonce i pro malé děti. Pro stroje jsou však problémy počítačového vidění nelehkou úlohou, která je v posledních letech předmětem zájmu čím dál většího počtu studií a výzkumů [5].

Lidé vnímají elektromagnetické záření v takzvaném viditelném spektru. Toto záření dopadá skrze čočku na sítnici lidského oka. Díky dvěma očím je možné vytvářet vjem třídimenzionálního světa. Lidé díky svému zraku, pokud je ze zdravotní stránky v pořádku, dokážou snadno pochopit a popsat prostředí, ve kterém se nachází. V mnoha případech je možné díky lidskému zraku velice snadno rozpoznat jednotlivé objekty, odlišit popředí od pozadí. Odborníci se po několik dekád snaží pochopit, jak funguje systém lidského zraku, pořád je však v této oblasti mnoho otázek nezodpovězených [42].

Výzkumníci specializující se na oblast počítačového vidění vyvíjejí systémy podobné lidskému zraku, které jsou schopny díky kameře nebo fotoaparátu zachytit scénu v podobě digitálního obrazu. Současné technologie umožňují díky matematice a sofistikovanému softwaru například na základě dostatečně velkého souboru dat spočítat přesný 3D model reálné scény, detekovat objekty a určit konkrétní hranici mezi nimi, sledovat objekt pohybující se v komplexním prostředí nebo dokonce rozpoznat konkrétní osoby podle jejich obličejů a také identifikovat účastníky silničního provozu pomocí metod hlubokého učení [42].

2.1 Technologické nároky počítačového vidění

V [9] autor popisuje počítačové vidění jako obor, který používá statistické metody k rozklíčování informace z dat pomocí modelů vytvořených na základě geometrie, fyziky a teorie učení. Tato disciplína vyžaduje znalosti několika technologií, bez kterých by nebylo možné pořídit data nebo z nich následně extrahovat potřebné informace:

1) Porozumění kamerám/fotoaparátům a fyzickému procesu pořizování videí a fotografií
2) Získání informace z hodnot jednotlivých pixelů
3) Zkombinování extrahovaných informací z jednotlivých fotografií a utvoření celku
4) Seskupování pixelů do skupin za účelem odvozování tvarů
5) Rozpoznávání objektů na základě geometrického tvaru nebo pravděpodobnosti

2.2 Aplikace počítačového vidění

Počítačové vidění jako takové má širokou škálu uplatnění. V [9] je autor rozděluje na aplikace staršího typu jako je navigace mobilních robotů, inspekce ve výrobě a v průmyslu obecně, zpracování satelitních snímků, špionáž a jiné vojenské aplikace. Dále zmiňuje aplikace novějšího typu, mezi které řadí interakci počítačů s lidmi, vyhledávání obrázků v digitálních knihovnách, analýzu lékařských snímků nebo realistické vykreslování umělých scén v počítačové grafice. Z [42] je převzat přehled několika dalších aplikací:

- **Optické rozpoznávání znaků (OCR):** konvertování textu z tištěné do elektronické podoby pro výstupy z laserových, inkoustových a jehličkových tiskáren nebo také knihtisku. Automatické rozpoznávání poznávacích značek automobilů.
- **Inspekce:** rychlá a spolehlivá inspekce dílů ve výrobě k zajištění kvality díky stereo kamerám a speciálnímu osvětlení. Hledání defektů pomocí X-ray detektoru.
- **Vytváření 3D modelů:** plně automatizovaná tvorba 3D modelů z leteckých snímků. Rekonstrukce 3D scény z fotografií, videií, souřadnic kamery. Robotické sbírání náhodně na sobě umístěných dílů v libovolné krabici díky stereo kameře (například brněnská firma Sanezoo [38]).
- **Autonomní vozidla:** detekce neočekávaných objektů jako jsou lidé na vozovce za podmínek, kdy radary a lidary nejsou funkční. Parkovací asistent, kamerové systémy s úhlem vidění 360° - aplikace transformací, zobrazení okolí vozidla "shora"(například firmy Mercedes Benz, Tesla).
- **Zábavní průmysl:** CGI (computer-generated imagery) sloučení počítačem vygenerovaného obsahu s natočeným videem díky sledování funkčních bodů. Použito například při natáčení filmu Hobit (drak Šmak).
- **Monitorování:** sledování vetřelců jako součást bezpečnostního systému střežených objektů. Analýza provozu na dálnici, měření rychlosti jedoucích aut v obcích. Monitorování bazénů kvůli tonoucím se plavcům.
- **Rozpoznávání otisků prstů a biometrická zařízení:** automatická autentifikace pro udělení přístupu do mobilních telefonů nebo počítačů na základě systému rozpoznávání obličejů, forenzní analýza.
- **Stabilizace videí:** využívání odhadů pohybu v roztřeseném videu za účelem stabilizace/odstranění třesu.
- **Měření mechanických vlastností materiálů:** extensometry k měření změn rozměrů materiálu a výpočtu napětí. 2D a 3D DIC (digital image correlation) systémy pro analýzu namáhání (například brněnská firma Xsight [45]).
- **Medicína:** technologie počítačového vidění a hlubokého učení nachází uplatnění v medicínských oborech jako je kardiologie (monitorování dat z ultra-

zvuku a echokardiogramu u akutních případů), patologie (vizuální inspekce vzorků pod mikroskopem), dermatologie (klasifikace zhoubné a nezhoubné kožní léze) [8], detekce diabetické retinopatie (soutěž o nejlepší model na Kaggle [19]).

3 Hluboké učení

Hluboké učení umožňuje výpočetním modelům (neuronovým sítím) složeným z mnoha výpočetních vrstev naučit se určitou reprezentaci dat s více úrovněmi abstrakcí. Metody hlubokého učení rapidně zlepšily současné technologie v oblasti rozpoznávání řeči, vizuálních objektů, detekce objektů a mnoho dalších. Pomocí algoritmu *zpětného šíření (backpropagation)* neuronové sítě mění své vnitřní parametry tak, aby minimalizovaly chybu. Hluboké konvoluční sítě přinesly průlom ve zpracování obrázků, videí, zvuku a rekuretní sítě v oblasti sekvenčních data jako je tex nebo řeč [22].

Konvenční techniky strojového učení byly limitovány ve své schopnosti zpracovat surová data jako jsou jednotlivé hodnoty pixelů fotografie. Pro práci s takovými daty je používán matematický model zvaný extraktor příznaků, který dokáže z dat vypočítat vhodnou reprezentaci nebo *vektor příznaků (feature vector)*. Z těchto příznaků pak další subsystém jako je například klasifikátor dokáže detekovat vzory ve vstupu. Do nedávné doby zkonstruování takového systému vyžadovalo pečlivou inženýrskou práci a vysokou úroveň expertíz, dnes se stroje z velkého množství dat mohou samy naučit extrahovat příznaky potřebné k detekci nebo klasifikaci [22].

3.1 Konvoluční neuronové sítě

Konvoluční sítě jsou navrženy tak, aby dokázaly zpracovat data v podobě vektorů nebo vícerozměrných polí, například barevný obrázek obsahující hodnoty pixelů ve třech barevných kanálech (RGB). Mnoho dat je reprezentováno v podobě vektorů/polí [22]:

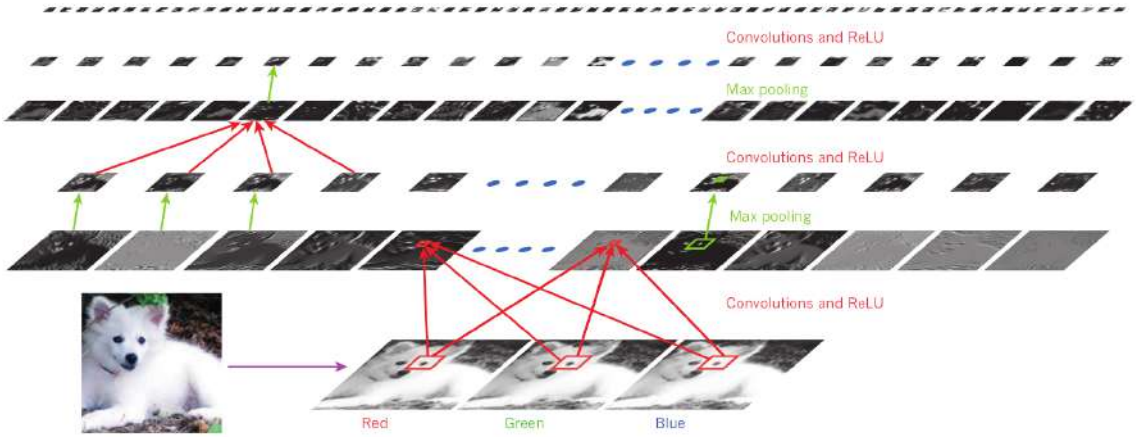
- **1D:** signály, sekvence (včetně jazyků)
- **2D:** obrázky, audio spektrogramy
- **3D:** videa, 3D modely

Architektura obecné konvoluční sítě (obrázek 1) je uspořádána do vrstev, z nichž prvních několik je jsou *konvoluční* a *pooling* vrstvy. Výstup z těchto vrstev bývá často přepočítáván aktivační funkcí (například *softmax* nebo *relu*). *Pooling* vrstvy obecně fungují jako rámeček o velikosti $N \times N$ pixelů, který přejíždí přes fotografii a z hodnot pixelů, které právě překrývá vypočítá výstup. Tento výstup je pak dále zpracováván. Příklady základních *pooling vrstev* [22]:

- **Max-pooling:** výstup vrstvy je maximální hodnota v rámečku
- **Average-pooling:** výstupem je střední hodnota čísel v rámečku

Obvykle se používají konvoluční bloky tvořené z konvoluční vrstvy, aktivační funkce a pooling vrstvy. Klasifikační sítě mívají několik takovýchto bloků řazených za sebou, výstup z nich jsou extrahované příznaky, které zpracovávají plně propojené

vrstvy. Zpětné šíření gradientu v konvolučních sítích funguje totožně jako u klasických, plně propojených neuronových sítí, což umožňuje měnit hodnoty konvolučních filtrů. Tento proces je nazýván učením [22].



Obr. 1: Ukázka konvoluční neuronové sítě. Ve spodní vrstvě obrázku lze vidět vstupní fotografii se třemi barevnými kanály. Horizontální vrstvy reprezentují výstupy jednotlivých vrstev sítě. Každý čtvercovitý obrázek ve vrstvě je vektor příznaků [22].

3.2 Funkce

V této kapitole budou stručně uvedeny některé typy funkcí používané pro hluboké učení. Pozornost bude soustředěna hlavně na funkce relevantní k práci.

3.2.1 Softmax

Softmax (rovnice (1)) je často využívaná funkce v posledních vrstvách klasifikačních sítí. Na vstupu přijímá vektor čísel $x \in \mathbf{R}$ a na výstupu dává $x \in [0, 1]$, suma všech čísel výstupního vektoru je rovna jedné, a proto je vhodné využít funkci pro přepočet výstupů sítě na hdonoty reprezentující pravděpodobnost [14].

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{1}$$

3.2.2 Sigmoida

V hlubokém učení je používána speciální verze logistické funkce zvaná *sigmoida* (rovnice (2)) [21].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

3.2.3 ReLu

ReLu (Rectified Linear Units) (rovnice (3)) je nelineární funkce často používaná jako aktivační funkce v konvolučních neuronových sítích. Podle autora v [21] je neuronová síť s *ReLu* aktivační funkcí mnohonásobně rychleji natrénovaná, než při využití *sigmoidy*.

$$f(x) = \max(0, x) \tag{3}$$

3.2.4 Leaky ReLu

Leaky ReLu (rovnice (4)) je vylepšením *ReLu*. Sítě využívající tuto funkci bývají méně náchylné k problému mizejícího gradientu. Konstanta a je volitelná (před trénováním), například v [33] autoři zvolili hodnotu $a = 0,1$.

$$\phi(x) = \begin{cases} x & \text{pro } x > 0 \\ ax & \text{ostatní} \end{cases} \tag{4}$$

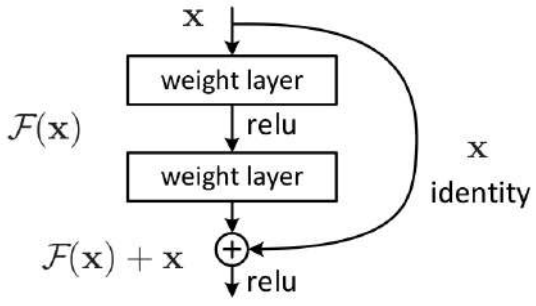
- a je volitelný parametr udávající skon křivky v záporné části osy x

3.3 Reziduální bloky

Hluboké neuronové sítě se skládají z několika milionů parametrů, a proto je obtížné je natrénovat. V [16] autoři představují tzv. *reziduální blok*, který umožňuje vytvářet a trénovat hlubší sítě.

Autoři adresují problém trénování hlubokých neuronových sítí, kdy s rostoucí hloubkou sítě přestává růst dosahovaná přesnost a v určitém bodě začne dokonce klesat [16].

Pokud by existovaly dvě stejné sítě a na jednu z nich byly přidány další vrstvy, které by jen mapovaly výstupní hodnoty společné části, vznikne hlubší síť se stejnou přesností. Hlubší síť by tedy měla mít schopnost být méně nebo minimálně stejně chybná jako mělčí. V [16] byl tento problém vyřešen návrhem tzv. *reziduálního bloku*, jeho schéma je na obrázku 2.

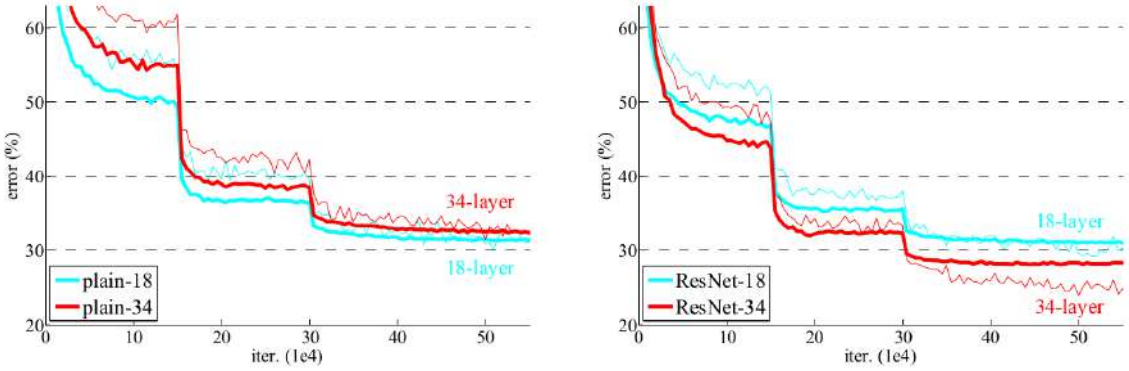


Obr. 2: Schéma reziduálního bloku [16].

Rovnice (5) popisuje reziduální blok:

$$y = \mathcal{F}(x, \{W_i\}) + x \tag{5}$$

- x vstupní vektor bloku
- y výstupní vektor bloku
- $\mathcal{F}(x, \{W_i\})$ je funkce reprezentující výstup reziduálního bloku



Obr. 3: Srovnání reziduální a klasické konvoluční sítě na datasetu *ImageNet* [16].

Obrázek 3 znázorňuje rozdíl v trénovací i validační chybě pro dvě různě hluboké sítě, každá hloubka je sestavena s i bez reziduálních bloků. Tenké křivky reprezentují trénovací chybu, tlusté validační. V levém grafu lze vidět průběh trénování klasických sítí, hlubší síť (34 vrstev) dosahuje v daném intervalu horších výsledků $(28,54\%)$ než síť s 18 vrstvami $(27,94\%)$. Sítě s reziduálními bloky dosahují lepších výsledků. Reziduální síť s 18 vrstvami dosáhla validační chyby $27,88\%$, zatímco hlubší síť $25,03\%$ [16].

3.4 Batch normalizace

Jeden z důvodů, proč je trénování hlubokých neuronových sítí obtížné, je měnící se rozdělení vstupů každé z vrstev (navíc se při trénování mění i parametry sítě). Tento fakt vyžaduje nižší *koeficient učení (learning rate)*, lepší inicializaci vah, také ztěžuje učení sítí používajících aktivační funkce s nasycením (například *ReLu*). Tento problém je v [18] nazýván *vnitřní posun kovarace (internal covariate shift)*.

Autoři v [18] navrhli metodu normalizace zvanou *Batch Normalization*, která řeší výše zmíněný problém, může být součástí architektury sítě jako jedna z vrstev a rapidně zvyšuje rychlost tréninku hlubokých neuronových sítí.

V rovnici (6) je vztah popisující normalizaci vstupního n-dimenzionálního vektoru $x = (x^{(1)}, ..., x^{(n)})$ [18]:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}] + \epsilon}} \tag{6}$$

- x je vstupní vektor
- $\hat{x}$ je znormalizovaný výstupní vektor
- $E[x^{(k)}]$ střední hodnota batche
- $Var[x^{(k)}]$ variance batche
- ϵ konstanta zajišťující numerickou stabilitu (obvykle malé číslo jako 10^{-6})

Autoři dále zmiňují, že znormalizovat vstup vrstvy může změnit její reprezentaci a uvádějí příklad normalizace vstupu *sigmoidy*, což by mohlo zapříčinit omezení funkce pouze na její lineární část. Jako řešení byly přidány dva parametry zajišťující, že výstup se může rovnat vstupu (při konkrétních hodnotách) [18].

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)} \tag{7}$$

- $\gamma^{(k)}$ váhy - naučitelný parametr
- $\beta^{(k)}$ bias - naučitelný parametr
- $y^{(k)}$ výstup *Batch Normalization* vrstvy

Pokud budou v rovnici (6) parametry $\gamma^{(k)} = \sqrt{Var[x^{(k)}] + \epsilon}$ a $\beta^{(k)} = E[x^{(k)}]$, bude výstup vrstvy totožný s jejím vstupem. Pokud by tedy bylo optimální, aby *Batch Normalization* vrstva plnila funkci identity, je možné toho tímto způsobem dosáhnout [18].

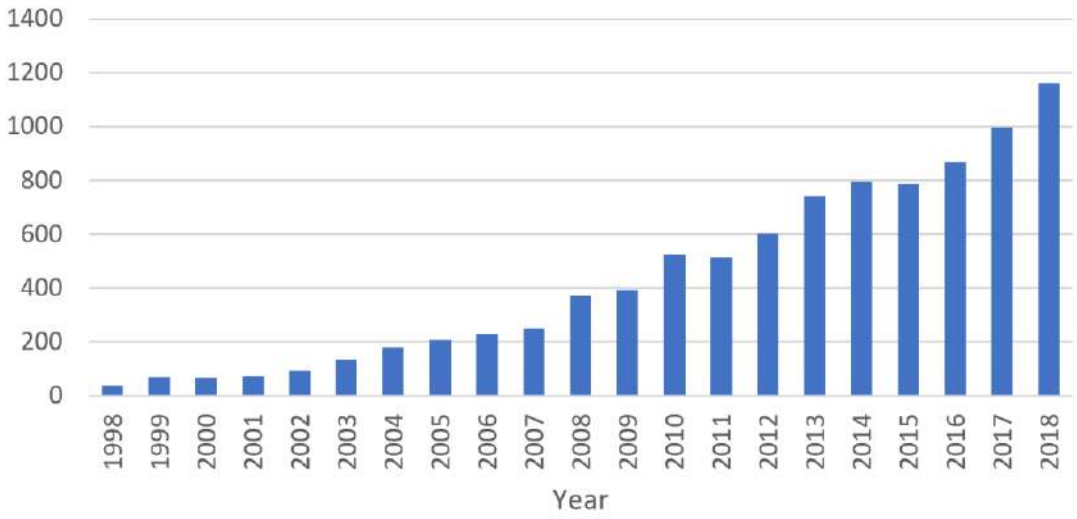
Metoda normalizace batche umožňuje nastavit pro trénování vyšší koeficient učení a větší benevolenci při inicializaci vah. Díky *Batch Normalization* se autorům [18] podařilo natrénovat klasifikační síť na datasetu *ImageNet* čtrnáctkrát rychleji na stejnou přesnost jako síť bez těchto vrstev.

3.5 Dropout

Pokud je hluboká neuronová síť trénována na malém množství trénovacích dat, dosahuje obvykle špatných výsledků přesnosti na testovací množině. V [17] je popsána technika, jak tento jev zvaný *overfitting* zmírnit náhodným vynecháváním tzv. *dropoutem*. Je určitá pravděpodobnost, například $p = 0,5$, při které neuron dá na svém výstupu hodnotu 0, tzn. polovina neuronů ve vrstvě je deaktivována a nepřispívá v dopředné ani zpětném šíření. Při inferenci se dropout nepoužívá a všechny neurony jsou aktivní [17].

4 Detekce objektů

Detekce objektů je jedním z nejzákladnějších a nejnáročnějších problémů v počítačovém vidění, hlubkém učení, a umělé inteligenci, kterému se v posledních letech dostává stále více pozornosti. Tento trend je možné pozorovat i na obrázku 4. Vývoj technologií pro detekci objektů v jednadvacátém století lze považovat za důležitý milník v oblasti počítačového vidění [47].



Obr. 4: Zvyšující se počet publikací na téma detekce objektů od roku 1998 do 2018 [47].

Rychlý vývoj technik hlubokého učení v minulých letech [22] umožnil pokrok v detekci objektů vedoucí k novým průlomovým algoritmům, díky jejichž přesnosti a rychlosti lze vidět stále častější nasazování těchto technologií do reálných aplikací [47].

4.1 Zařazení detekce objektů do identifikačních úloh

Problém identifikace účastníku silničního provozu je možné řešit několika způsoby. Použití algoritmů pro detekci objektů je jedna z možností. Pro lepší náhled na toto téma bude ukázán stručný výčet identifikačních úloh podle [5]:

- **Klasifikace objektů:** řadí objekt na fotografii do jedné z kategorií (určení zda objekt na fotce je člověk, auto, pes..).
- **Identifikace objektů:** přesnější určení typu daného objektu (určení rasy psa na fotce).
- **Zaměření objektu:** zaměření objektu a určení jeho polohy.

- **Detekce orientačních bodů objektu:** určení klíčových bodů objektů na fotografii (pro rozpoznávání obličejů, gest).
- **Detekce objektů:** klasifickace objektů a zaměření (určení jejich polohy ve fotografii).
- **Segmentace:** přiřazení pixelů v obraze jednotlivým objektům.

Autor v [5] dále konstatuje, že úlohy optického rozpoznávání znaků, klasifikace obrázků, detekce objektů, detekce obličejů a rozpoznávání obličejů společně patří do skupiny **rozpoznávání objektů**. Výše zmíněné techniky, stejně jako většina technik hlubokého učení jsou v počítačovém vidění použity pro detekci objektů, segmentaci nebo nějakou formu detekce. To znamená zaměřit daný objekt v obraze rámečkem, získat z obrazu určitou sekvenci symbolů nebo přiřadit každý jeden pixel ve fotce objektu, ke kterému náleží [14].

4.2 Cíle detekce

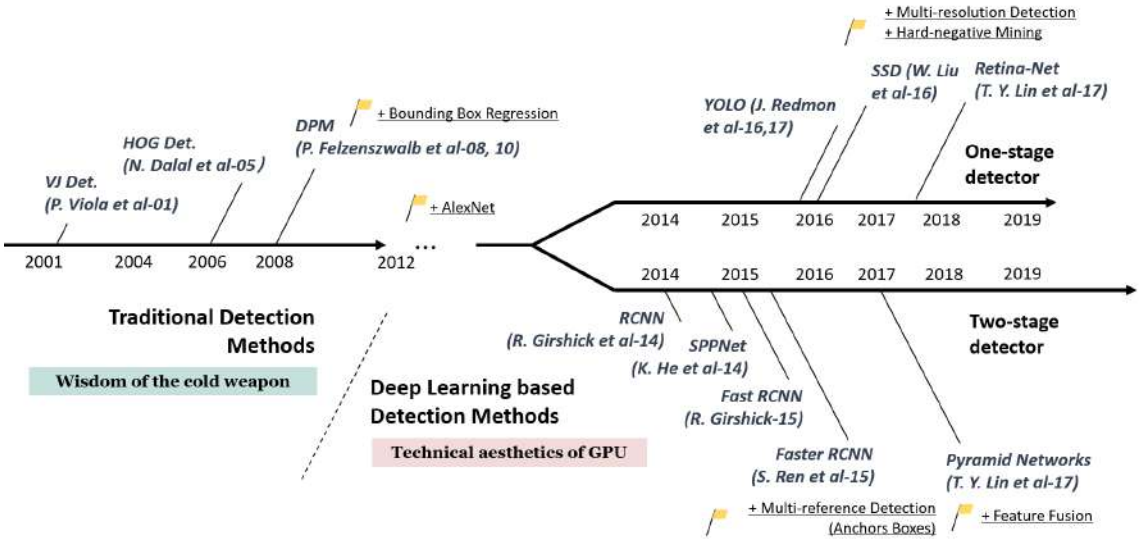
Úloha detekování objektů je velmi důležitou součástí počítačového vidění, která se zabývá zaměřováním vizuálních objektů patřících do určité klasifikační třídy (například lidé, auta, zvířata atd.). Cílem detekce objektů je vyvinout výpočetní modely a techniky, které jsou schopny z digitálního obrazu extrahovat nejzákladnější informaci potřebnou pro počítačové vidění - *kde se které objekty nachází?* [47].

Schopnost detekovat objekty je důležitou prerekvizitou pro komplexnější úkoly počítačového vidění jako je sledování cíle, detekce náhlé události, analýza chování a sémantické porozumění scény. Účelem je zaměřit cíl zájmu v digitálním obraze, přesně vyhodnotit jeho klasifickační třídu a přiřadit mu ohraničující rámeček, tzv. *bounding box*, jak lze vidět na obrázku 5 [7].

Obr. 5: Růžový rámeček, tzv. *bounding box*, fotka z datasetu MSCoco 2017 [25].

4.3 Dělení detekčních algoritmů

Obrázek 6 znázorňuje vývoj algoritmů pro detekci objektů. Algoritmy před rokem 2012 jako *Viola Jones Detectors*, *HOG Detector* a *Deformable Part-based Model (DPM)* patří do skupiny **tradičních detekčních metod** a nazývají se *Wisdom of the cold weapon era*. Tyto algoritmy nejsou založené na technikách hlubokého učení. Kvůli nedostatku dat a omezenému výpočetnímu výkonu bylo nutné ručně navrhnout sofistikovaný systém extrakce příznaků [47].



Obr. 6: Graf znázorňující vývoj detekčních algoritmů od roku 2001 [47].

Rok 2012 je považován za milník vývoje detekčních algoritmů. S rozvojem výpočetní techniky se na scénu vrátily konvoluční sítě, které nyní byly schopny naučit se robustní extrakci příznaků z digitálního obrazu. V [21] byl představen AlexNet, konvoluční neuronová síť trénovaná na GPU (avšak ne první) na datasetu ImageNet. Využívala více konvolučních jader k extrakci příznaků, dropout a ReLu vrstvy pro rychlejší konvergenci při tréninku [46]. Později vyvinuté detekční algoritmy využívají převážně konvoluční vrstvy, vznikla tak skupina **detekčních metod založených na hlubokém učení**, tyto využívají převážně GPU pro trénování [47]. Tyto algoritmy se dělí na dvě hlavní skupiny [41]:

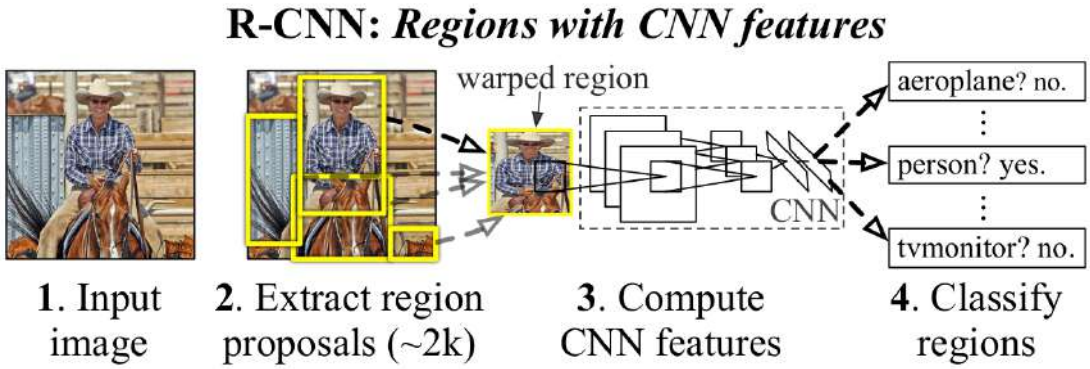
1. Two-stage detektory
2. Single-stage detektory

4.4 Two-stage detektory

Two-stage detekrory používají v první fázi detekce *Region Proposal Network* k určení místa zájmu ve fotografii a předají tato místa klasifikátoru, který ve druhé fázi provede regresi ohraničujícího rámečku a určí klasifikační třídu. Tyto modely dosahují **vyšší přesnosti**, ale při inferenci jsou většinou **pomalejší** [41].

4.4.1 RCNN

V roce 2014 Ross Girshick představil v [13] konvoluční neuronovou síť s názvem *RCNN (Region based convolutional neural network)*. Autor v článku uvádí přesnost (mAP) $53,7\%$ na datasetu PASCAL VOC 2010.



Obr. 7: Architekrura sítě RCNN [13].

Na obrázku 7 popisující architekruru sítě lze vidět, že algoritmus na vstupním obrázku nejprve zaměří oblasti zájmu tzv. *region proposal*. Těchto oblastí je přibližně dva tisíce. Každá z extrahovaných oblastí zájmu je přeškálovaná do dané velikosti a následně předána konvoluční síti k extrakci příznaků. Nakonec natrénovaný regresní model provede predikci třídy a ohraničujícího rámečku v rámci daného regionu [47].

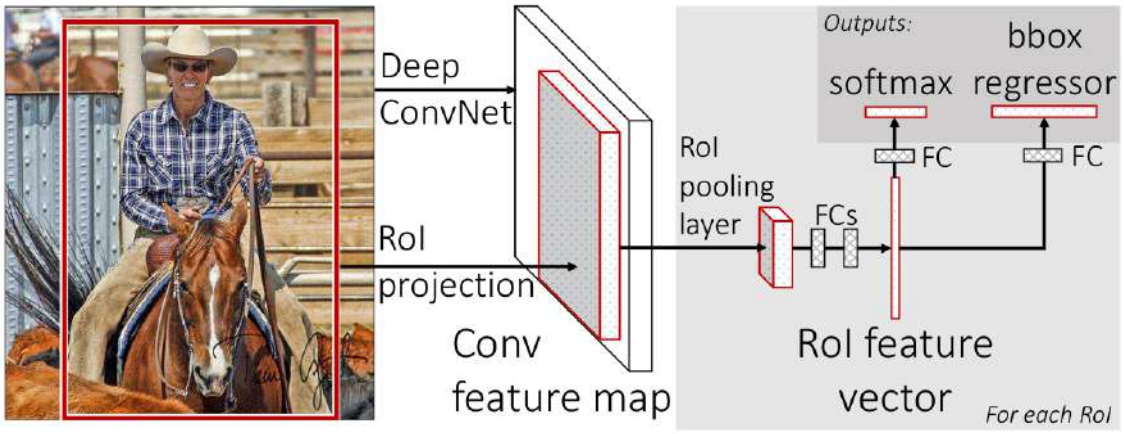
Přesnost sítě *RCNN* je ve srovnání s tradičními metodami vyšší, ale systém má zjevné nevýhody. Počet operací je velmi vysoký (dva tisíce oblastí zájmů pro jednu fotku) a efektivita nízká (klasifikují se i oblasti, kde žádný objekt není). Každá z oblastí zájmu je také přeškálována na pevně danou velikost, což může vést ke zkreslení objektu [7].

4.4.2 Fast RCNN

Autor *RCNN* v [12] popsal vylepšený algoritmus s názvem *Fast RCNN*. Na datasetech VOC2007 a VOC2012 ve svém textu uvádí přesnost (mAP) $70,0\%$. Změny oproti původnímu algoritmu [7]:

- Pro klasifikaci je použita funkce *softmax*.

- Využívá *pyramid pooling vrstvy.*
- Poslední vrstva klasifikátoru je nahrazena dvěma plně projenými vrstvami.



Obr. 8: Architekrura sítě Fast RCNN [12].

Obrázek 8 znázorňuje fungování sítě. Vstup je zpracován konvolučními vrstvami, které extrahují příznaky. Podobně jako předchůdce vybere oblasti zájmu. Každá z oblastí zájmu je pak zpracována *region of iterest pooling vrstvou* a výstup je napojen na dvě plně propojené vrstvy, které spočítají vektor příznaků dané oblasti zájmu (region of iterest feature vector). Ten následně vstupuje (zvlášť) do dvou plně propojených vrstev. Funkce *softmax* z výstupu první vrstvy vypočítá zařazení do klasifikačních tříd. Z druhé vrstvy vypočítá regresor souřadnice ohraničujícího rámečku [12].

4.4.3 Faster RCNN

Třetí verze původního *RCNN* algoritmu byla představena v [36] ve stejném roce jako jeho předchůdce (2015). Přesnost sítě na datasetech:

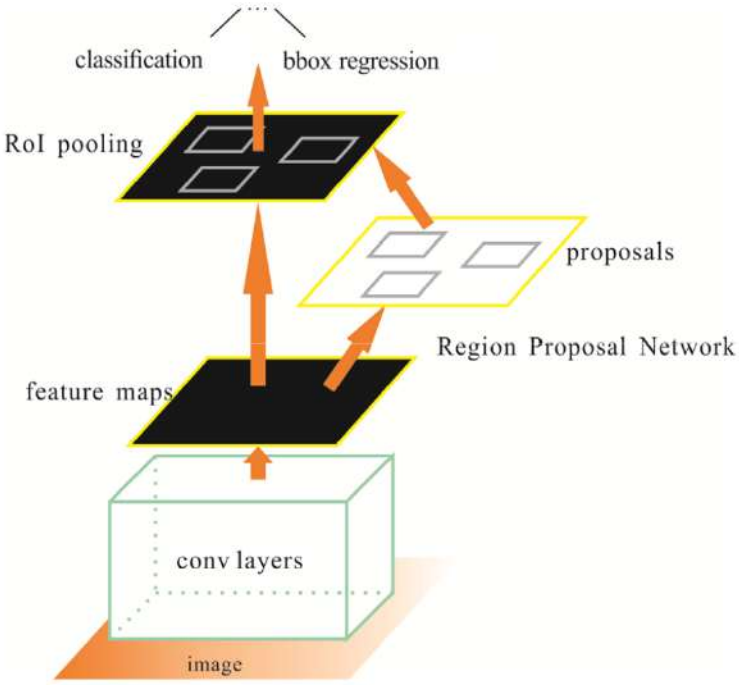
- **COCO** (mAP@0,5=42, 7%, mAP@[0,5;0,95]=21, 9%)
- **VOC2007** (mAP=73, 2%)
- **VOC2012** (mAP=70, 4%)

Využívá sítě zvané *RPN (region proposal network)* (konvoluční neuronová síť) k výběru oblastí zájmu [36]. Svojí rychlostí, jak napovídá název studie *Faster RCNN: Towards Real-Time Object Detection with Region Proposal Networks*, se blíží k použití v reálném čase, pořád však ve výpočtu existuje určitá redundance [47]. Autoři algoritmu v článku zmiňují rychlost $17FPS$ (snímků za sekundu). Změna oproti *Fast RCNN* spočívá také v integraci jednotlivých bloků (extrakce příznaků), generování oblastí zájmu, regrese ohraničujících rámečků, klasifikace atd.) do jednoho celku [47].

Region proposal network

RPN - Region proposal network je síť, která na vstupu přijímá obrázek libovolného rozměru. Výstup této sítě obsahuje set oblastí zájmů ve tvaru obdélníku, z nichž je každá ohodnocená tzv. *skóre objektovosti (objectness score)* (pravděpodobnost, že oblast zájmu patří do jedné ze skupinky tříd nebo náleží pozadí) [36].

Jak lze vidět na obrázku 9, fotografie na vstupu projde konvoluční sítí (extraktorem příznaků). Příznaky vstupují do *RPN*, ta vygeneruje oblasti zájmu, které pak společně s příznaky zpracovává *region of interest pooling vrstva*. Výstupem sítě jsou klasifikace a souřadnice ohraničujících rámečků [7].



Obr. 9: Architekrura sítě Faster RCNN [7].

4.4.4 Feature pyramid networks

V roce 2017 v [23] byla představena koncepce sítě zvaná *FPN (Feature pyramid network)*. Autoři ji popisují jako základní komponentu rozpoznávacích systémů pro detekci v různých škálách, zároveň ale zmiňují, že tato technika v dosud vyvinutých sítích nebývá aplikována kvůli velké výpočetní náročnosti.

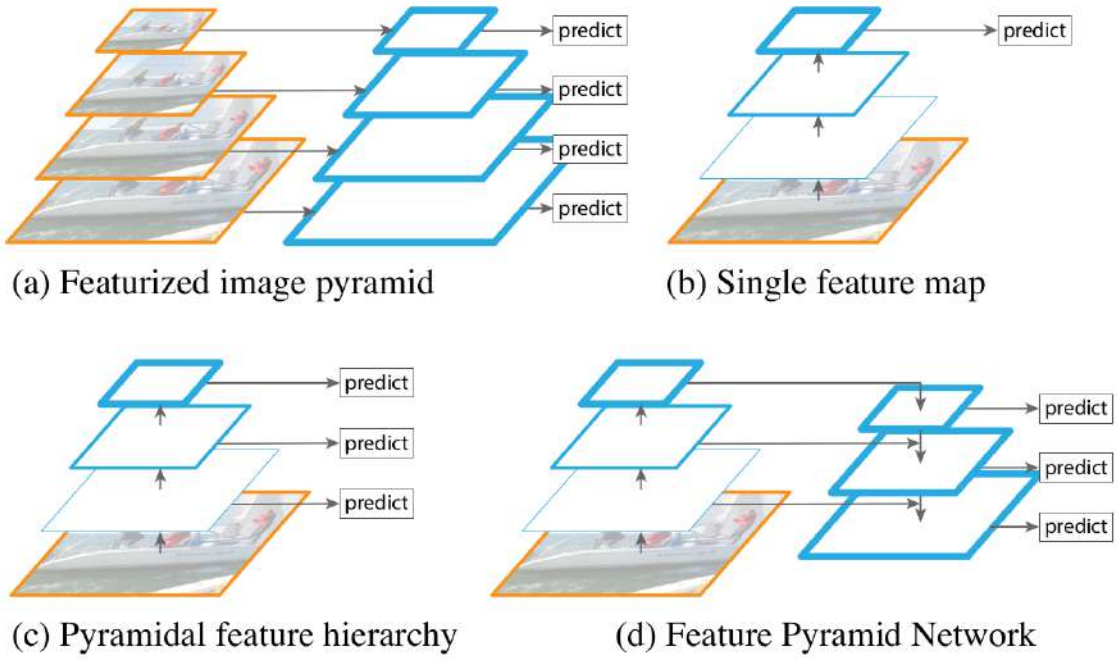
Na obrázku 10 autoři srovnávají jednotlivé typy pyramidových architektur a zvažují jejich výhody a nevýhody [23]:

(a) **Featurized image pyramid:** používají se pyramidy obrázků k vytvoření pyramidy příznaků, z každého patra této pyramidy se provadí predikce. Příznaky jsou vypočítány pro každou škálu zvlášť, a proto je tento způsob pomalý.

(b) **Single feature map:** model počítající predikce z jedné škály. Studie zmiňuje, že většina detekčních systémů v té době je založena na tomto principu.

(c) **Pyramidal feature network:** jedna z možností je využít hierarchii pyramidových příznaků spočítaných konvolučními vrstvami při snižování rozlišení podobně jako u modelu (a), s tím rozdílem, že pyramida příznaků je spočítána z jedné fotky.

(d) **Feature Pyramid Network:** model navrhnutý autory, rychlostí srovnatelný s modely (b) a (c), ale přesnější, tento model je níže rozebrán.



Obr. 10: Srovnání architektur pyramidových sítí. Extrahované příznaky jsou označeny modrým obrysem, čím silnější obrys, tím sémanticky bohatší příznaky [23].

FPN - Cesta zdola nahoru

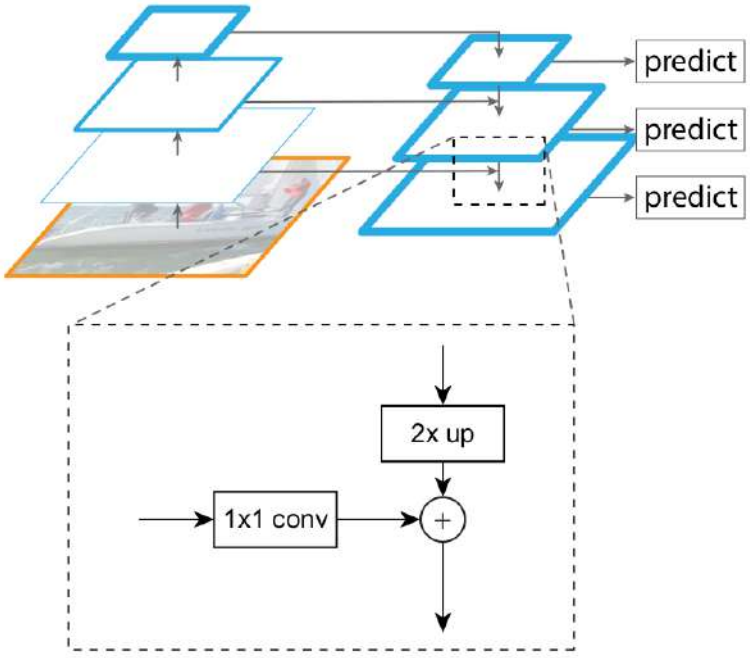
Cesta zdola nahoru je důsledkem operací prováděných konvolučními sítěmi na vstupním obrázku (tzv. *forward pass*). Data na výstupu jsou nazývána příznaky. V konvolučních sítích je na sebe často napojeno několik vrstev, jejichž výstupy mají stejné rozlišení příznaků, tyto vrstvy označujeme jako vstvy ve stejné fázi sítě. Každá úroveň pyramidy odpovídá jedné fázi. Poslední vrstva každé fáze má nejbohatší příznaky, tudíž je vybrána jako referenční a na této budou prováděny další operace. Na obrázku 11 je tato část zobrazena na levé straně (zdola nahoru) [23].

FPN - cesta shora dolů a boční spojení

Na vrcholu pyramidy jsou extrahované příznaky s nižším rozlišením a bohatší sémantikou. Tyto příznaky jsou následně při cestě shora dolů (pravá část) převzorkovány

na vyšší rozlišení a obohaceny připojením příznaků z levé pyramidy (zdola nahoru). Připojené příznaky mají vždy stejné rozlišení jako přivzorkované [23].

Na obrázku 11 je zobrazen blok znázorňující převzorkování na vyšší úroveň (nadvzorkování) faktorem 2. Pro jednoduchost operace nadvzorkování byla použita metoda nejbližšího souseda. Připojené příznaky jsou navíc ještě zpracovány konvoluční vrstvou s konvolučním jádrem o velikosti 1×1 kvůli zredukování počtu kanálů [23].



Obr. 11: FPN blok znázorňující cesty zdola nahoru, shora dolů a boční spojení [23].

4.5 Single-stage detektory

Single-stage detektory přistupují k problému detekce objektů jako k regresnímu problému. Ze vstupního obrázku a anotací se přímo učí určit třídu, pravděpodobnost a souřadnice ohraničujícího rámečku. Tyto modely obvykle dosahují **nižší přesnosti**, ale zato jsou mnohem **rychlejší** než two-stage detektory, a proto mohou být použité pro aplikace v reálném čase [41].

4.5.1 Yolo v.1

V roce 2016 byl Josephem Redmonem v [33] představen detekční algoritmus s názvem *Yolo - You Only Look Once* schopný běžet v reálném čase. Přehled rychlostí a přesností:

- Rychlejší verze tohoto algoritmu dokáže běžet při $155 FPS$ na datasetech **VOC 2007+2012** s přesností (mAP=$52,7\%$)

- Přesnější real-time verze s rychlostí $45FPS$ na datasetech **VOC 2007+2012** a přesností (mAP=$63,4\%$)
- Nejpřesnější verze, která ale neběží v reálném čase - $21FPS$ na datasetech **VOC 2007+2012** s přesností (mAP=$66,4\%$)

Tento způsob přistupuje k detekci objektů jako k regresnímu problému, kde jediná konvoluční neuronová síť provádí predikci ohraničujícího rámečku a pravděpodobnosti třídy přímo z obrázku. Algoritmus rozdělí fotografii na několik buněk (do mřížky). Buňka, ve které leží střed objektu je zodpovědná za jeho predikci [24].

První verze algoritmu *Yolo* je sice rychlá, ale ztrácí na konkureční (v té době existující) algoritmy v přesnosti. Objekty ve fotografii dokáže rychle zaměřit, s těmi menšími má ovšem lokalizační problémy [33]. Více v kapitole 5.

4.5.2 RetinaNet

Rychlé a z hlediska architektury jednoduché single-stage sítě nedosahovaly takových přesností jako two-stage sítě. V roce 2017 se autoři [24] na tento problém zaměřili a prozkoumali ho. Studie zmiňuje, že hlavní příčinou je tzv. *nevyváženost popředí a pozadí*.

- Za **popředí** se považují buňky (obraz rozdělený do mřížky podobně jako u *Yolo v.1*), které obsahují objekt patřící do jedné z klasifikačních tříd.
- Za **pozadí** se považují buňky, které neobsahují žádný objekt.

Tento problém nesouvisí s nevyvážeností tříd v datasetu, ale s podstatou fungování single-stage detektorů objektů. Fotografie rozdělená na mřížku o velikosti $S \times S$ obsahující jeden klasifikační objekt naučí predikovat buňku (jednu), která je zodpovědná za detekci toho objektu. Ostatní buňky ($S \times S - 1$) se z této fotografie učí predikovat pozadí. Je zřejmé, že buňky se učí predikovat především pozadí, protože jeho plocha v datasetech (většinou) dominuje [24].

Focal Loss

Pro řešení tohoto problému autoři navrhli kriteriální funkci zvanou *focal loss*, která by měla zmírnit extrémní nevyváženost popředí a pozadí při učení single-stage detektorů [24].

$$BCE(p, y) = -[y \cdot \log(p) + (1 - y) \cdot \log(1 - p)] \tag{8}$$

- $p \in [0, 1]$ a je předpovězená pravděpodobnost výskytu objektu
- $y \in \{0, 1\}$ a reprezentuje přítomnost objektu (z anotace)

Autoři pro jednoduchost vychází z rovnice (8) pro *binární křížovou entropii*, vše platí i pro *křížovou entropii*. Ze vztahu (8) lze odvodit vztahy (9) a (10) [24].

$$BCE(p, y) = \begin{cases} -\log(p) & \text{pro } y = 1 \\ -\log(1 - p) & \text{pro } y = 0 \end{cases} \tag{9}$$

$$p_t = \begin{cases} p & \text{pro } y = 1 \\ 1 - p & \text{pro } y = 0 \end{cases} \tag{10}$$

Na základě výše uvedených výrazů lze vyjádřit *binární křížovou entropii* jako v rovnici 11 [24].

$$BCE(p_t) = -\alpha_t \cdot \log(p_t) \tag{11}$$

- α_t je faktor vyvažující význam popředí a pozadí. Tuto hodnotu lze spočítat analogicky s p_t v rovnici (10).

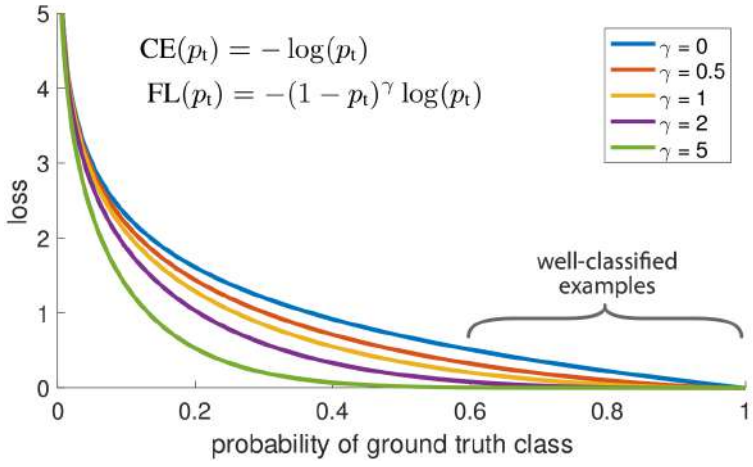
Rovnice (11) popisuje modifikaci *binární křížové entropie* pro vyvážení vlivu popředí a pozadí na kriteriální funkci. V praxi by mohla být hodnota α nastavená jako inverzní hodnota frekvence výskytu třídy (pokud je třída t zastoupena n-krát, hodnota $\alpha_t = 1/n$). Přidání vah kompenzuje nevyvážené zastoupení popředí a pozadí, ale nerozlišuje mezi snadnými a obtížnými detekcemi [24].

- **Snadné** jsou detekce, které síť úspěšně klasifikovala jako popředí nebo pozadí.
- **Obtížné** jsou detekce, které síť klasifikovala špatně.

$$FL(p_t) = -\alpha_t \cdot (1 - p_t)^{\gamma} \cdot \log(p_t) \tag{12}$$

- $\gamma \in [0, 5]$, tzv. *parametr ostření*, čím vyšší, tím víc se síť soustředí na trénovaní obtížných objektů
- $(1 - p_t)^{\gamma}$, tzv. *modulační faktor*

Reprezentaci *focal loss* funkce lze vidět v rovnici (12). Oproti vážené *binární křížové entropii* je zde přidán tzv. modulační faktor [24].



Obr. 12: Závislost hodnoty kriteriální funkce na predikované pravděpodobnosti [24].

Pokud je predikce špatná a p_t nízké, modulační faktor se blíží 1 a kriteriální funkce není ovlivněna. Při hodnotě p_t blížící se 1 a modulační faktor bude skoro

nulový, tak se hodnota ztrátové funkce pro dobře klasifikovaný objekt sníží, ukazuje obrázek 12. Nastavení hodnoty parametru ostření na 0 znamená, že funkce je totožná s *křížovou entropií*. Hondota kriteriální funkce *focal loss* je z většiny tvořena špatně klasifikovanými objekty, a proto se na ně klade větší důraz při zpětném šíření gradientu [24].

Architektura

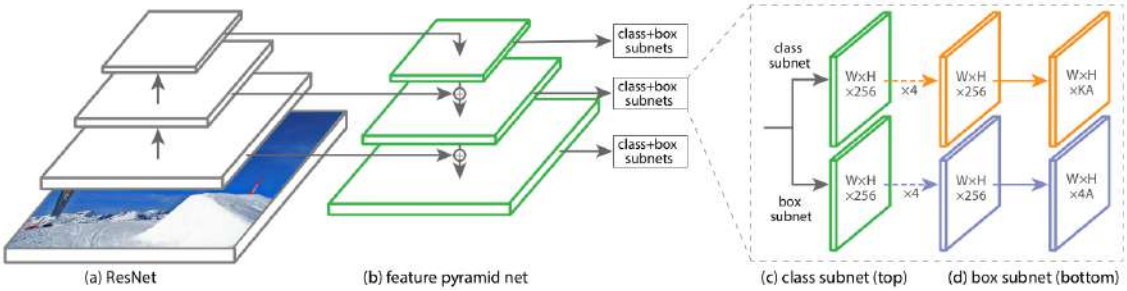
Na obrázku 13 lze vidět architekru one-stage sítě RetinaNet [24]:

(a) **ResNet:** Páteř celé sítě tvoří extraktor příznaků zvaný *ResNet*, ten generuje bohatou, vícestupňovou pyramidu příznaků svými dopřednými konvolučními operacemi [24].

(b) **Feature Pyramid Net:** Nejvyšší patro pyramidy je nadvzorkováno a z příslušných pater pyramidy příznaků *ResNetu* jsou k němu tyto příznaky připojeny. RetinaNet takto získá predikce ze tří různých rozlišení [24].

(c) **Klasifikační podsíť:** Každá predikce je rozdělena na několik regionů (mřížka o rozměrech $W \times H$). Na každou predikci je napojena klasifikační podsíť vypočítávající pravděpodobnost přítomnosti objektu v každém z regionů. Klasifikační podsíť je malá plně konvoluční síť (FCN - Fully convolutional network). Parametry této podsítě jsou sdíleny všemi vrstvami pyramidy. Každé z predikovaných rozlišení na výstupu vypočítá $W \times H \times KA$ predikcí (K - počet klasifikačních tříd, A - počet kotev). [24].

(d) **Regresní podsíť:** Podobně jako u klasifikační podsítě je na každé z predikovaných rozlišení připojena regresní podsíť. *RetinaNet* využívá tzv. kotvy (anchors boxy). Regresní část se snaží vypočítat odsazení kotvy od reálného ovjektu (pokud nějaký existuje). Každé z predikovaných rozlišení na výstupu vypočítá $W \times H \times 4A$ predikcí (4 - počet čísel potřebných pro určení ohraničujícího rámečku, A - počet kotev). [24].



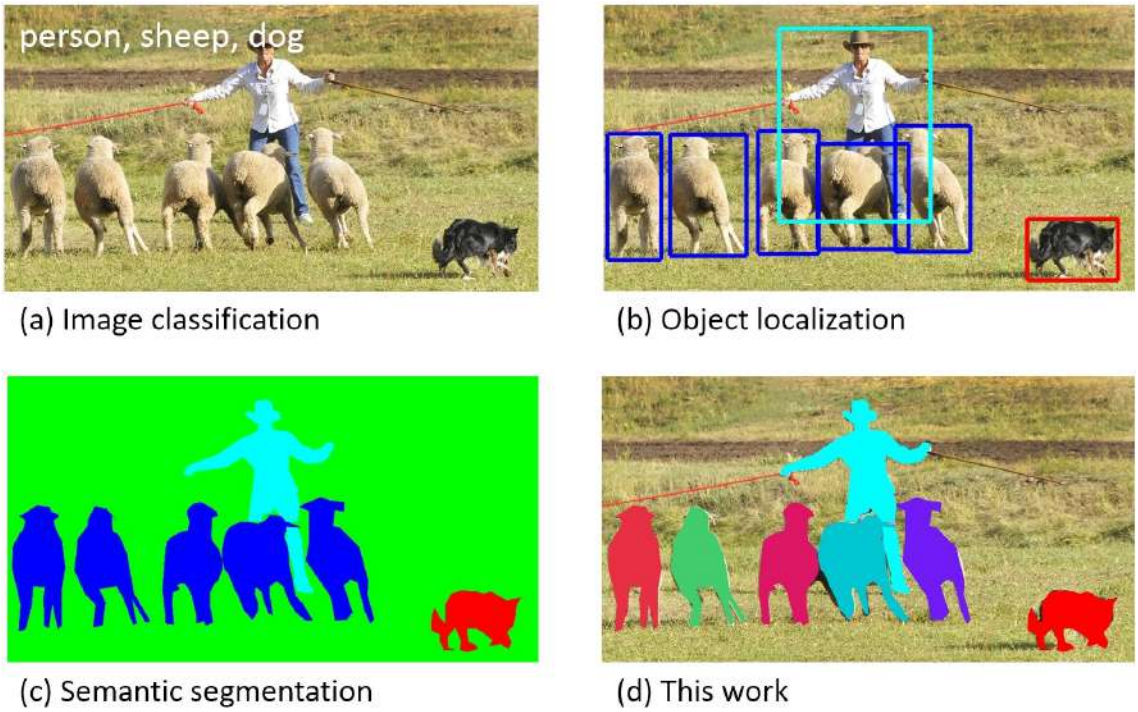
Obr. 13: Architektura sítě RetinaNet [24]

.

Autoři zmiňují, že klasifikační a regresní konvoluční neurovnová síť mají podobnou strukturu, ale nesdílí vnitřní parametry [24].

4.6 Datasety

Vytváření větších datasetů fotek s menší zaujatostí je klíčové pro vývoj robustních systémů počítačového vidění. V posledních dvou dekádách bylo vydáno několik datasetů pro detekci objektů [47]. Některé z nich tato práce níže rozebere.



Obr. 14: Ukázka anotací datasetu MSCoco [26]

.

Pro lepší názornost jsou na obrázku 14 ukázány jednotlivé typy anotací, ze kterých se mohou učit různé architektury sítí určené k různým účelům [26].

(a) **Klasifikace obrázku:** Prosté zařazaní objektů do vybraných klasifikačních tříd.

(b) **Detekce objektů:** Klasifikace objektů a jejich lokalizace pomocí ohraničujících rámečků.

(c) **Sémantická segmentace:** Segmentace obrázku na úrovni pixelů (přiřazení každého pixelu konkrétní klasifikační třídě).

(d) **MSCoco:** Segmentace jednotlivých instancí objektů.

Výčet nejznámějších datasetů a stručný popis podle [10, 47], datasety PASCAL VOC a MC-Coco budou podrobněji rozebrány v kapitolách níže.

4.6.1 ILSVRC (ImageNet Large Scale Visual Recognition Challenge)

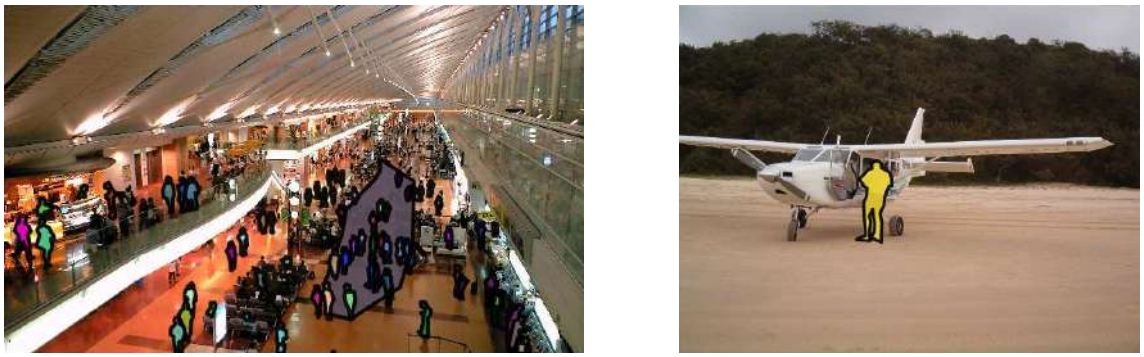
Dataset [37] obsahuje 517 tisíc obrázků a 534 tisíc anotací (o dva řády více než VOC dataset) [47]. V [10] autor u ukázkového obrázku 15 upozorňuje, že objekty datasetu

jsou velké a situované do středu. Na tomto datasetu byl natrénován například model konvoluční neuronové sítě AlexNet [21].

Obr. 15: Ukázky obrázků z datasetu ImageNet [10]

4.6.2 SUN (Scene Understanding)

Tento dataset (ukázka na obrázku 16) poskytl pro výzkum a vývoj komplexní sbírku anotovaných obrázků zahrnující širokou škálu scén. Obsahuje necelé čtyři a půl tisíce kategorií a přes tři sta tisíc segmentací jednotlivých instancí objektů ve sto třiceti tisících obrázcích [10].

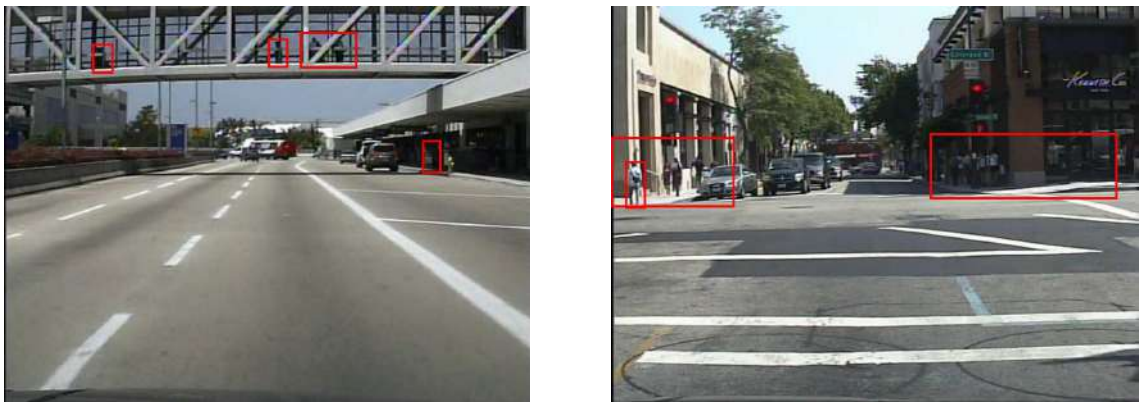


Obr. 16: Ukázky obrázků z datasetu SUN [10]

4.6.3 Caltech Pedestrian Dataset (Caltech)

Dataset představený v roce 2012 obsahuje zhruba 10 hodin záznamu a rozlišení 600×400 pořízený při 30 snímcích za sekundu. Záznam je pořízen z vozidla projíždějícího městem (obrázek 17). Anotace zahrnují všechny chodce na každém snímku.

Jsou vytvořeny dva druhy ohraničujících rámečků, *plný* a *viditelný*. *Plný* ohraničující rámeček těsně ohraničuje celého chodce, pokud je část chodce na snímku



Obr. 17: Ukázky obrázků z datasetu Caltech [10]

skrytá, odhaduje se velikost schované části. *Viditelný* ohraničující rámeček na druhou stranu ohraničuje pouze část chodce, která není zakryta žádným jiným objektem. Tento přístup je rozdílný například od VOC datasetu, který označuje pouze viditelné části objektů. Caltech dataset obsahuje necelých tři a půl tisíce ohraničujících rámečků na dvě stě padesáti tisících snímcích [10].

4.6.4 OID (Open Images Detection)

Představeno v roce 2018, formátem podobné jako dataset *MS-Coco*, ale rozsahem mnohem větší. Tento dataset se zaměřuje na dva hlavní úkoly. První je standardní detekce objektů a druhý je detekce vizuálních vztahů mezi jednotlivými objekty [47].

4.6.5 PASCAL VOC (Visual Object Classes)

V letech 2005 až 2012 probíhaly každoroční výzvy *PASCAL VOC*, které jsou považované za jedny z nejdůležitejších soutěží v počátcích vzniku komunity kolem počítačového vidění a hlubokého učení spojeného s detekcí objektů [10].

Obr. 18: Ukázky obrázků z datasetu VOC [10]

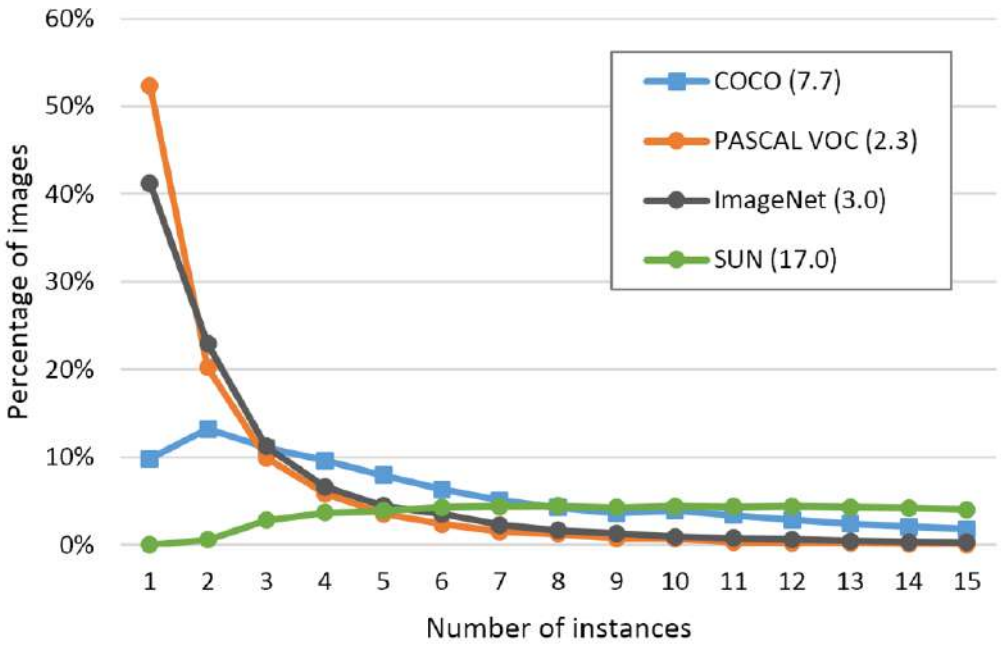
Na ukázce (obrázek 18) jsou viditelné nedostatky datasetu *VOC*, né všechny osoby jsou oanotovány. Na levém obrázku chybí cyklista v pozadí, na pravém člověk nastupující do vlaku, přitom menší osoba v pozadí na nástupišti označená je. Dataset obsahuje 20 tříd a vyzývá ke splnění několika úkolů týkající se klasifikace, detekce objektů, sémantické segmentace a detekce události. V roce 2009 se stal populárním datasetem pro srovnávání detekčních a jiných algoritmů v komunitě počítačového vidění. Jeho nejznámější verze jsou **VOC2007** a **VOC2012** [47].

4.6.6 MS-Coco (Common Objects in Context)

Dataset *MS-Coco* je v současnosti nejnáročnější dostupný dataset pro detekci objektů. Podobně jako u *VOC* probíhá každoroční soutěž od roku 2015. Ve srovnání s *ILSVRC* má *Coco* méně kategorií, ale zato více instancí. Verze **MS-Coco 2017** obsahuje 80 kategorií, sto šedesát čtyři tisíc obrázků a osm set devadesát sedm tisíc anotací. Největší progres *Coco* datasetu je, že vedle ohraničujících rámečků obsahuje také segmentaci pro každou instanci objektu, což přispívá k přesnější lokalizaci. Početné zastoupení mají také malé objekty s plochou menší než 1% celkové plochy obrázku. Hustota objektů na obrázcích datasetu je vyšší než u *VOC* a *ILSVRC*, což lze vidět na obrázku 19 [47].

4.6.7 Srovnání

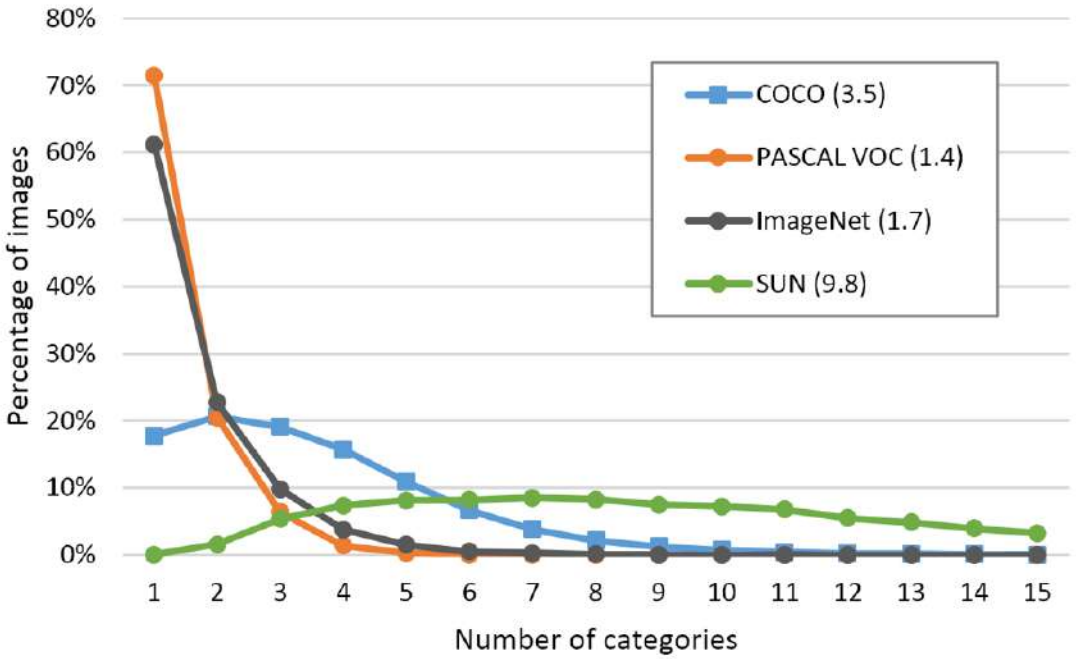
Autoři v [26] porovnávají některé z populárních datasetů, ty se liší ve velikosti, klasifikačních třídách i v typech fotografií.



Obr. 19: Graf srovnání počtu instací objektů na jednu fotografii [26]

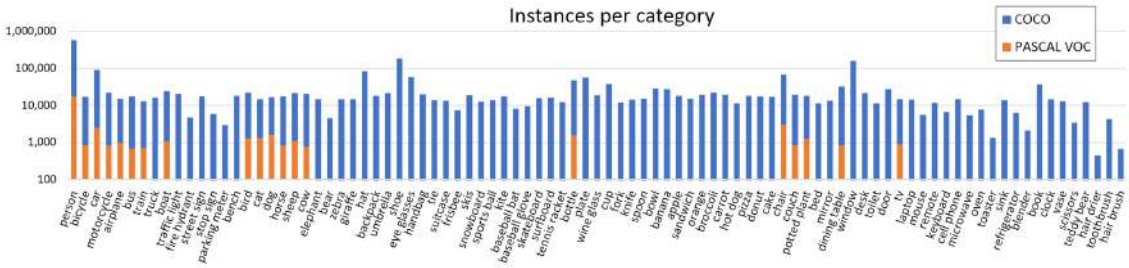
.

Graf na obrázku 19 srovnává hustotu instací objektů na fotografiích datasetů, tzn. kolik objektů je průměrně na jednom obrázku. *ImageNet* byl vytvořen, aby zachytil velké množství různých kategorií objektů. *SUN* je zaměřen na prostředí a objekty, které se ve skutečnosti spolu běžně vyskytují. Primární aplikace datasetu *PASCAL VOC* je detekce objektů na fotografiích s přirozenou kompozicí, takové fotografie zachycují scény běžné v každodenním životě. *MS-Coco* dataset je podobně jako *VOC* navrhnutý pro detekci a segmentaci objektů vyskytujících se v jejich přirozeném prostředí, odtud název *běžné objekty v kontextu* [26].



Obr. 20: Graf srovnání počtu klasifikačních tříd na jednu fotografii [26].

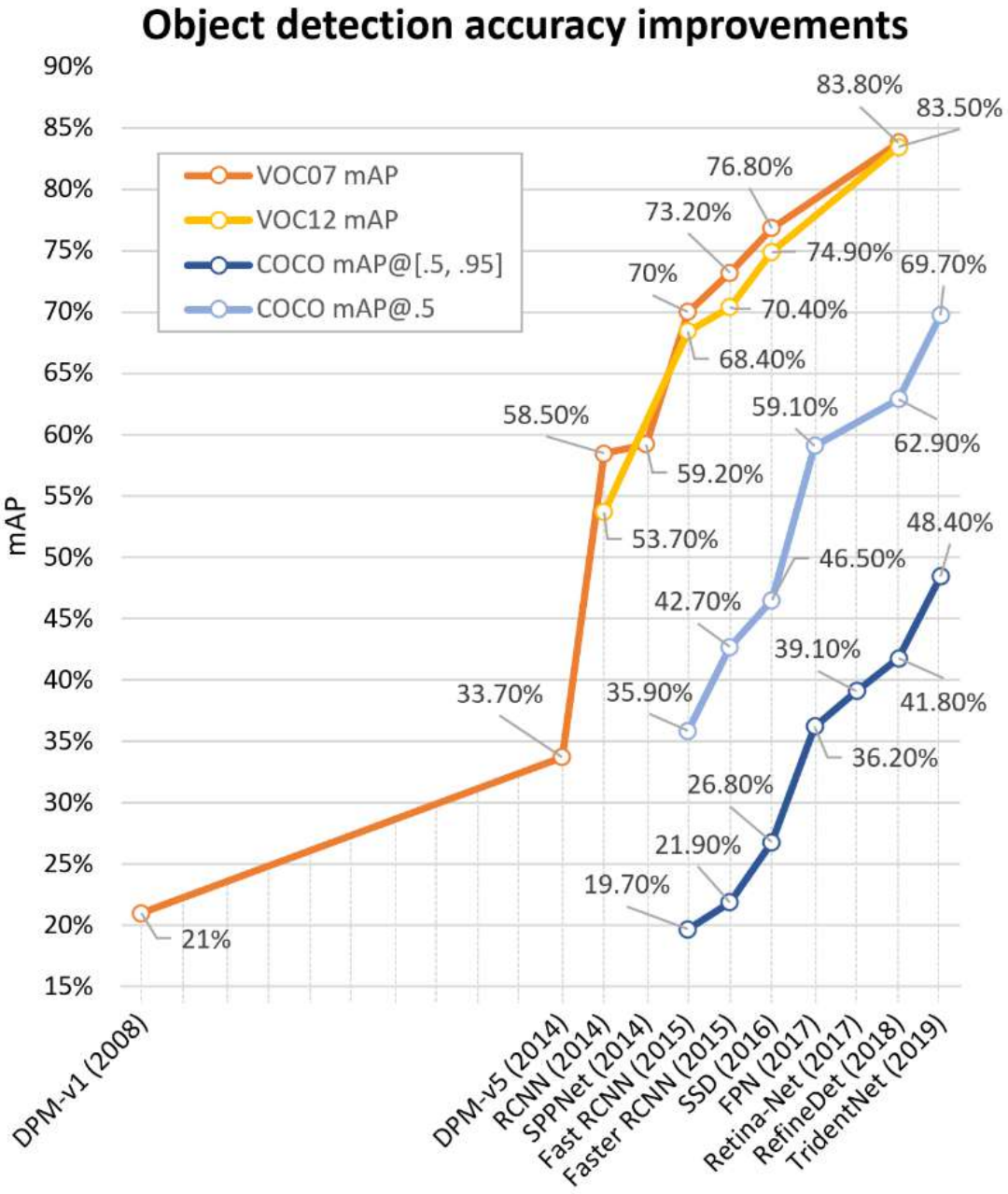
Obrázek 20 v grafu srovnává počet kategorií (klasifikačních tříd) na jednu fotografii. Většina fotografií datasetů *VOC* a *ImageNet* obsahuje jednu nebo dvě třídy (viz průměr). Nejvíce kategorií na jednu fotografii pokytuje dataset *SUN*. *MS-Coco* obsahuje průměrně 3,5 klasifikačních tříd na jeden obrázek [26].



Obr. 21: Graf srovnání počtu instací konkrétních tříd datasetů *VOC* a *Coco* [26].

MS-Coco dataset je bohatší co se týče počtu klasifikačních tříd (80) proti *PASCAL VOC* (20). Graf na obrázku 21 také znázorňuje, že *MS-Coco* také překonává *VOC* v počtu instancí objektů na jednu třídu.

Graf na obrázku 22 zachycuje dosažené přesnosti detekčních algoritmů od roku 2008 na datasetech *PASCAL VOC* a *MS-Coco*. Z grafu je zřejmé, že dataset *MS-Coco* je pro detekční algoritmy větší výzvou, a proto v současnosti slouží jako referenční dataset k porovnávání přesností detekčních sítí [47].



Obr. 22: Dosažené přesnosti detekčních algoritmů od roku 2008 [47]

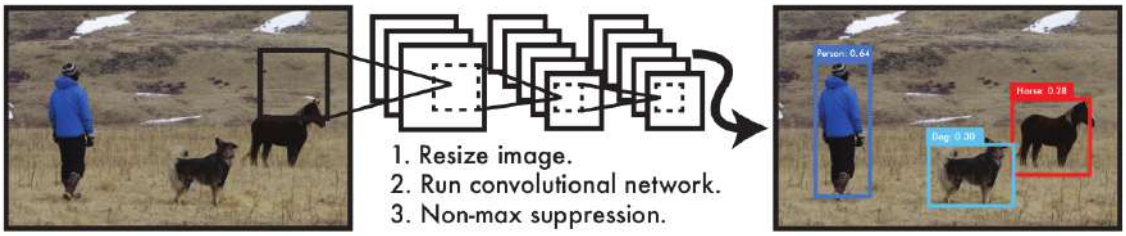
.

5 Yolo - You Only Look Once

Tato kapitola se bude věnovat detekčním algoritmům z rodiny *Yolo* představené Josephem Redmonem a dalšími v [4, 11, 33, 34, 35].

5.1 Yolo v.1

První verze algoritmu *Yolo* z [33] byla ve své době velkým pokrokem v oblasti detekce objektů. Tento single-stage detektor přistupuje k detekci objektů jako k regresnímu problému, z pixelů obrázku přímo počítá souřadnice ohraničujících rámečků a pravděpodobnosti výskytu klasifikačních tříd [33].



Obr. 23: Zobrazení principu fungování detekčního systému první verze sítě *Yolo* [33]

.

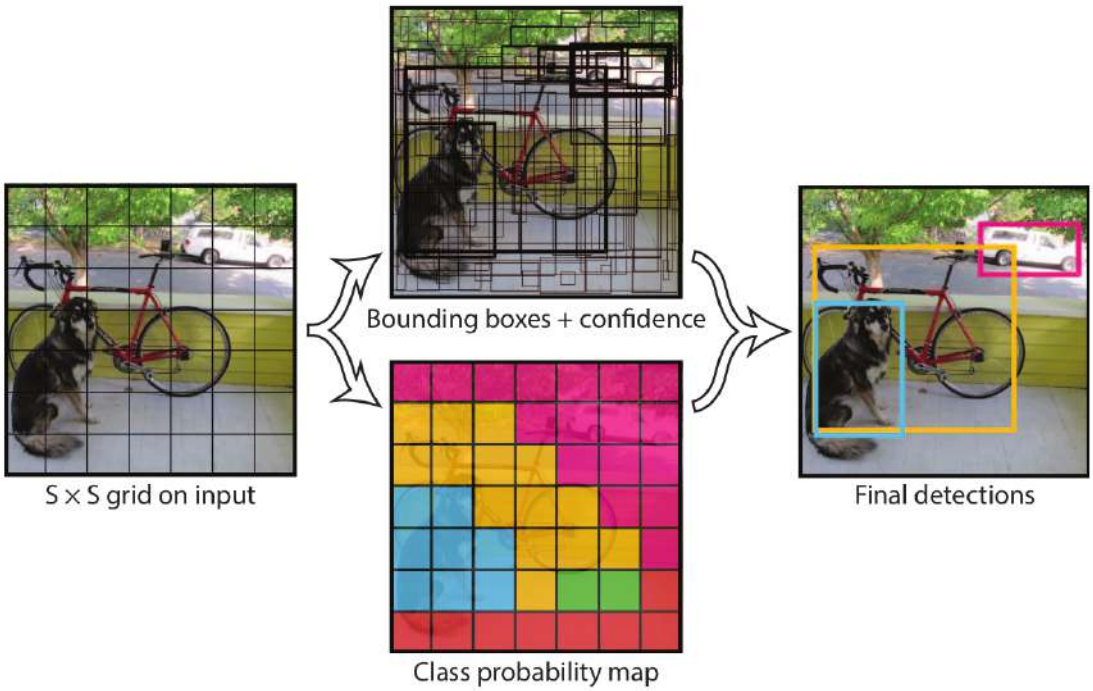
Obrázek 23 popisuje posloupnost kroků následovaných algoritmem při detekci objektů z fotografie [33]:

1. Původní velikost obrázku se změní na pevně danou hodnotu 448×448 pixelů
2. Jediná konvoluční síť provede výpočty na vstupní fotografii
3. Výsledkem je tenzor předem dané velikosti, který reprezentuje ohraničující rámečky. Né všechny jsou použitelné, a proto se většina z nich vytřídí algoritmem zvaným *Non-maximum supression.*

Algoritmus provádí konvoluční operace na celé fotografii (né jen na vybrané oblasti zájmu jako některé jiné detekční algoritmy). Vstup je rozdělen na mřížku o rozměrech $S \times S$, mřížka obsahující střed objektu je zodpovědná za jeho detekci. Každá buňka mřížky predikuje B ohraničujících rámečků a příslušnou pravděpodobnost přítomnosti objektu v daném rámečku [33].

Každý ohraničující rámeček se skládá z pěti údajů: *x, y, šířka, výška, pravděpodobnost.* Souřadnice *x, y* reprezentují střed ohraničujícího rámečku relativně k příslušné buňce, takže platí: $x, y \in [1, 0]$. *Šířka a výška* jsou relativní k obrázku, podobně jako pro *x, y* platí: $w, h \in [1, 0]$. Předpovězená *pravděpodobnost* reprezentuje hodnotu *IOU - intersection over union* mezi predikovaným a skutečným rámečkem. Každá buňka navíc predikuje C podmíněných pravděpodobností pro klasifikační třídy. Nevýhoda sítě je, že predikuje pouze jeden soubor podmíněných pravděpo-

dobností C (bez ohledu na počet predikovaných rámečků B), a tak jedna buňka může předpovědět maximálně jeden objekt.



Obr. 24: Model sítě ukazující rozdělení vstupní fotografie na mřížku, pravděpodobnostní mapu, všechny predikované ohraničující rámečky a finální detekce [33]

.

Síť rozdělí forografii na mřížku $S \times S$ (jako v obrázku 24), pro každou buňku vypočítá B ohraničujících rámečků a C podmíněných pravděpodobností. Na výstupu sítě se pak nachází tenzor o rozměrech $S \times S \times (B \cdot 5 + C)$. Autoři ve své práci zvolili pro dataset *PASCAL VOC* následující parametry [33]:

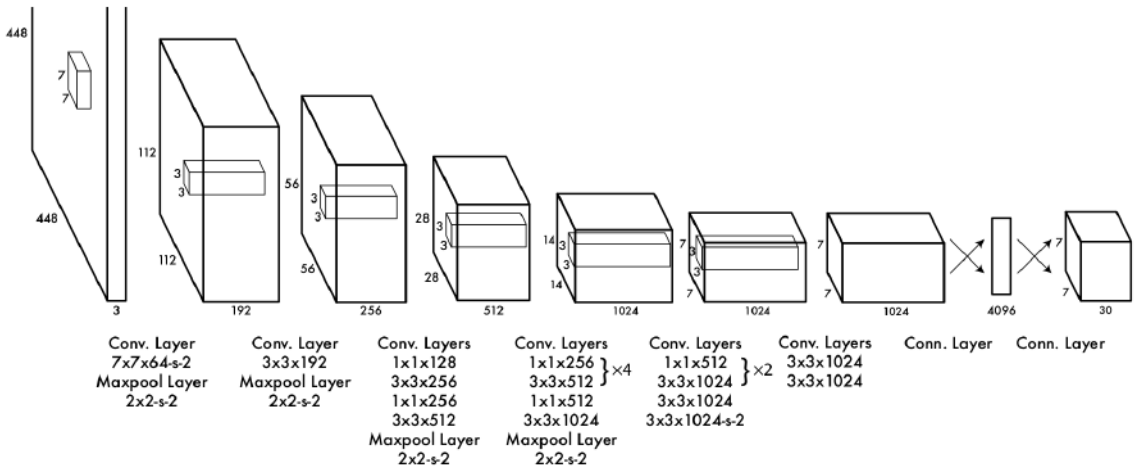
- $S = 7$
- $B = 2$
- $C = 20$

5.1.1 Architektura sítě

Model byl naimplementován jako konvoluční neuronová síť, trénink a testování proběhlo na datasetu *PASCAL VOC*. První část sítě, extraktor příznaků zvaný *darknet-19*, provede konvoluční operace a tím extrahuje příznaky. Na něj navázané plně propojené vrstvy následně predikují pravděpodobnosti a souřadnice. Síť se skládá z dvaceti čtyř konvolučních a dvou plně propojených vrstev. Síť používá jednoduchá konvoluční jádra s rozměrem 1×1 a na ně napojená další o velikosti 3×3. Celou architekturu sítě lze vidět na obrázku 25 [33].

Síť za konvolučními bloky extraktoru příznaků používá *max-pooling vrstvu*, za extraktorem přízků *average-pooling vrstvu*. Poslední vrstvu tvoří linerární aktivační funkce, ostatní konvoluční vrstvy používají *leaky-relu* funkci s parametrem 0,1, kterou popisuje rovnice (4) [33].

Mimo hlavní model byla také vytvořena menší a rychlejší verze této sítě za účelem posunutí hranice rychlosti detekčních algoritmů. Autoři ji nazývají *Fast Yolo*, oproti hlavní verzi má jen 9 konvolučních vrstev s míň konvolučními jádry. Kromě architektury se rychlá verze svými parametry nijak neliší od *Yolo* [33].



Obr. 25: Architektura první verze sítě *Yolo* [33]

.

5.1.2 Trénink

Síť byla předtrénována na datasetu *ImageNet* s tísícem klasifikačních tříd. Předtrénováno bylo prvních dvacet konvolučních vrstev z obrázku 25, na ně byla napojena *average-pooling vrstva* a plně propojená vrstva. Trénování této části trvalo jeden týden a na *ImageNet* dosáhlo přesnosti 88% [33].

Na předtrénovaný extraktor příznaků byly napojeny čtyři konvoluční vrstvy a dvě plně propojené vrstvy s náhodně inicializovanými vahami [33].

Celá síť byla následně trénována po 135 epoch na datasetu *PASCAL VOC 2007 a 2012*. Hyperparametry nastavené při trénování [33]:

- velikost batche = 64
- momentum = 0.9
- úbytek vah = 0.0005

Při trénování byla zvolena následující strategie. Nejprve bylo trénování inicializováno s parametrem *rychlosti učení (learning rate)* 10^{-3}, autor zmiňuje, že trénink je na začátku při vyšší hodnotě nestabilní a diverguje. Po ustálení se rychlost učení

zvýší na 10^{-2}, takto bylo trénováno po 75 epoch, následně sníženo na 10^{-3} po 30 epoch a posledních 30 epoch při rychlosti učení 10^{-4} [33].

Pro vyhnutí se přetrénování (tzv. *overfitting*) byl použit *dropout* a augmentace dat (náhodné translace a přeškálování až do 20% původní velikosti obrázku, dále náhodné úpravy expozice a saturace) [33].

5.1.3 Kriteriální funkce

Autoři nejprve vyzkoušeli optimalizovat součet kvadratických chyb mezi modelem a požadovným výsledkem. Tato metoda přiděluje stejnou váhu lokalizačním a klasifikačním chybám. Mnoho buněk ve fotce neobsahuje žádný objekt, a tak se více učí rozpoznávat pozadí než popředí (tento problém byl podrobněji rozebrán v kapitole 4.5.2). Pravděpodobnostní skóre buněk je tímto snižováno. Autoři tento způsob zavrhli, protože vede k nestabilitě modelu a způsobuje divergenci při rané fázi trénování [33].

Jako řešení byly vytvořeny konstanty λ_{coord} a λ_{noobj}. Konstanta λ_{coord} byla nastavena na hodnotu 5 a násobí výstup kriteriální funkce pro souřadnice ohraničujícího rámečku. Druhá konstanta λ_{noobj} byla nastavena na hodnotu 0,5 a násobí výstup kriteriální funkce pro pravděpodobnostní skóre buněk, ve kterých se nenachází žádný objekt. Tato operace modifikuje velikosti jednotlivých složek kriteriální funkce, ze kterých se následně počítá a do sítě zpětně propaguje gradient [33].

Další nedostatek původního návrhu kriteriální funkce je přiřazování stejných vah odchylkám šířky a výšky u velkých a malých ohraničujících rámečků. Malá odchylka u velkého rámečku by měla mít menší váhu než malá odchylka u malého rámečku (může způsobit velkou nepřenost). Jako částečné řešení problému autoři přistupují k predikované šířce a výšce jako ke druhé odmocnině opravdové šířky a výšky, tzn. výstup sítě pro výšku a šířku se musí před dalším použitím přepočítat [33].

Jak již bylo zmíněno, *Yolo* predikuje více ohraničujících rámečků pro jednu buňku. Při trénování je za predikci objetu zodpovědný vždy pouze jeden ohraničující rámeček (z celkového počtu B rámečků na jednu buňku) a to ten, který má největší hodnotu *intersection over union* se skutečným rámečkem. Každý z rámečků v buňce se tak specializuje na predikci objektů různých velikostí [33].

$$\theta_{xy} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \psi_{i,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \tag{13}$$

$$\theta_{wh} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \psi_{i,j}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \tag{14}$$

- θ_{xy} kvadratická chyba polohy středu rámečku
- θ_{wh} kvadratická chyba šířky a výšky rámečku

- $\psi_{i,j}^{obj} \in \{0,1\}$ a reprezentuje zodpovědnost ohraničujícího rámečku j v buňce i za predikci objektu
- x a $\hat{x}$ reprezentují odsazení skutečného a predikovaného středu rámečku v horizontálním směru od hranice příslušné buňky popořadě (stejné platí pro y a $\hat{y}$ ve vertikálním směru)
- w a $\hat{w}$ reprezentují šířku skutečného a predikovaného rámečku popořadě (stejné platí pro výšku h a $\hat{h}$)

Rovnice 13 a 14 reprezentují vztahy pro výpočet střední kvadratické chyby mezi šířkou, výškou a středem predikovaného a skutečného rámečku [33].

$$o_{obj} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \psi_{i,j}^{obj} \left(C_i - \hat{C}_i\right)^2 \tag{15}$$

$$o_{noobj} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \psi_{i,j}^{noobj} \left(C_i - \hat{C}_i\right)^2 \tag{16}$$

- o_{obj} kvadratická chyba predikce třídy pro popředí (buňka s objektem)
- o_{noobj} kvadratická chyba predikce třídy pro pozadí (buňka bez objektu)
- C_i a $\hat{C}_i$ reprezentují zakódování skutečných a predikovaných klasikačních tříd (popořadě)
- $\psi_{i,j}^{noobj} \in \{0,1\}$ a reprezentuje příslušnost ohraničujícího rámečku j v buňce i k pozadí

Rovnice (15) a (16) reprezentují vztahy pro výpočet střední kvadratické chyby predikcí klasifikačních tříd. Výpočet chyby pro pravděpodobnostní skóre vyjadřuje rovnice (17) [33]:

$$p = \sum_{i=0}^{S^2} \psi_i^{obj} \sum_{c \in classes} \left(p_i(c) - \hat{p}_i(c)\right)^2 \tag{17}$$

- $\psi_i^{obj} \in \{0,1\}$ a reprezentuje přítomnost objektu v buňce i
- $p_i(c)$ a $\hat{p}_i(c)$ reprezentují pravděpodobnostní skóre skutečného ($p_i(c) \in \{0,1\}$) a predikovaného ohraničujícího rámečku

Výsledná hodnota kriteriální funkce je dána vztahem (18) [33]:

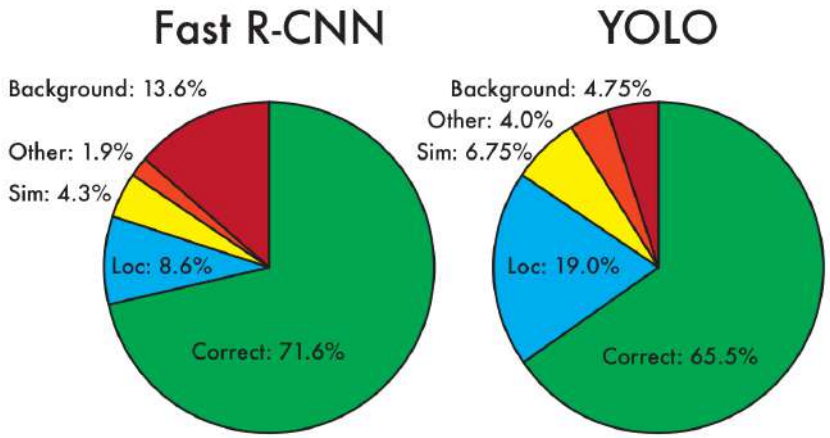
$$L = \lambda_{coord}(\theta_{xy} + \theta_{wh}) + o_{obj} + \lambda_{noobj} o_{noobj} + p \tag{18}$$

5.1.4 Omezení

První verze sítě *Yolo* má velká omezení co se týče rozlišení většího počtu objektů situovaných blízko k sobě. Je to způsobeno predikcí omezeného počtu ohraničujících rámečků pro jednu buňku, která navíc může identifikovat pouze jeden objekt [33].

Model se učí predikovat ohraničující rámečky na základě dostupných dat, proto může mít problémy při detekovaní objektů v neobvyklém kontextu nebo rozměru [33].

Jak bylo již zmíněno výše, kriteriální funkce přikládá stejnou váhu malým odchylkám u velkých i malých ohraničujících rámečků. Malá chyba u velkého rámečku nepředstavuje zásadní problém, zatímco u malého rámečku může mít velký vliv na hodnotu *intersection over union*. Obrázek 26 zobrazuje srovnání přesnosti a chyb algoritmů *Yolo* a *Fast RCNN*. Největší zdrojem chyb u první verze *Yolo* je špatná lokalizace ohraničujících rámečků [33].



Obr. 26: Srovnání chybovosti sítí *Yolo* a *Fast RCNN*. Největším zdrojem chyb u *Fast RCNN* je detekce pozadí, zatímco u *Yolo* jsou to lokalizace ohraničujících rámečků [33].

5.2 Yolo v.3

Tato kapitola bude pojednávat především o třetí verzi algoritmu *Yolo* [35]. Zahrnuty ovšem budou i některé body z druhé verze [34] a to hlavně ty, které jsou pro fungování třetí verze klíčové. Autor ve své studii [35] zmiňuje, že hlavní přínos práce je implementace několika nových věcí z oblasti hlubokého učení (týkající se počítačového vidění) a jejich spojení dohromady v nové síti tak, aby se maximalizovala přesnost mAP na datasetu *MS-Coco*. Byl vyvinut nový extraktor příznaků zvaný *darknet-53*, větší a přesnější než první verze (*darknet-19*) [35].

5.2.1 Kotvy

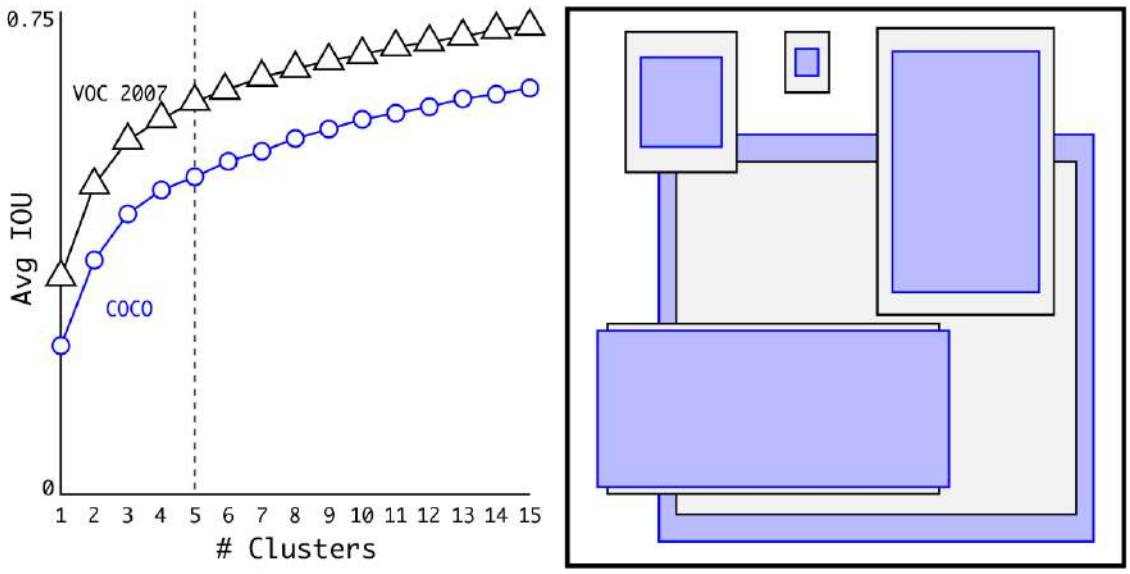
V [34] byla implementována koncepce kotev (tzv. *anchor boxů*), tuto metodu využívá i třetí verze. První verze algoritmu predikuje výšku a šířku ohraničujícího rámečku

přímo. Při využití kotev se pro síť proces zjednodušuje na predikci hodnoty, o kterou se má kotva zvětšit, aby vznikl rámeček ohraničující skutečný objekt [34].

Z první verze byly odstraněný plně propojené vrstvy, predikce ohraničujících rámečků zajišťují kotvy. Další změna se týká predikce tříd, pravděpodobnostní skóre i zařazení do klasifikačních tříd je nyní počítáno pro každou kotvu. To znamená, že síť umí narozdíl od první verze detekovat více objektů se středem v jedné buňce a tyto objekty mohou každý příslušet do jiné třídy. Počet objektů, který je maximálně buňka schopná predikovat se rovná počtu kotev sítě [34].

Autor konstatuje, že s využitím kotev se snížila přesnost sítě o $0,3\%$, ale zvýšila se schopnost sítě detekovat více objektů situovaných nahusto u sebe, protože se výrazně zvýšil počet predikovaných ohraničujících rámečků [34].

Volba rozměrů kotev byla nejprve vybrána ručně, ale pro lepší výsledky se autoři rozhodli využít algoritmus *k-means clustering* a na trénovací sadě datasetu *MS-Coco* vypočítat optimální rozměry [34].



Obr. 27: K-means shlukování pro výpočet kotev ohraničujících rámečků na datasetech *VOC* a *MS-Coco* [34]

Obrázek 27 (vpravo) znázorňuje rozměry kotev (ve srovnání s velikostí celého obrázku) vypočítaných pro oba datasety. Graf (vlevo) zobrazuje střední hodnotu *intersection over union* pro jednotlivé datasety a počet shluků [34].

Metrika zvolená pro algoritmus *k-means* je popsaná v rovnici (19). Při použití Euklidovské vzdálenosti by větší rámečky generovaly větší chybu. Použití *intersection over union* jako metriky zajišťuje nezávislost na rozměrech rámečků [34].

$$d(\text{rámeček, těžiště}) = 1 - IOU(\text{rámeček, těžiště}) \tag{19}$$

5.2.2 Ohraničující rámečky

Poloha ohraničujícího rámečku je počítána jako odsazení v ose x a y od referenčního bodu (u obrázků obvykle v levém horním rohu) konkrétní buňky. Alternativní způsob je učit síť predikovat odsazení středů rámečků relativně k celé fotografii, to však podle autorů vede k nestabilitě, hlavně v raných iteracích tréninku [34].

Pokud má být střed ráměčku uvnitř buňky zodpovědné za jeho predikci a hodnota reprezentující odsazení relativní k příslušné buňce, musí obecný výstup sítě pro osazení $x \in [0, 1]$. Výstupy sítě pro odsazení středů rámečků však nabývají nejrůznějších hodnot $x \in \mathbf{R}$. Pro omezení hodnot odsazení na požadovaný interval je použita *logistická funkce* v rovnici (2) [35].

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \end{aligned} \tag{20}$$

- $t_{x,y}$ reprezentuje výstup sítě pro osazení středu ohraničujícího rámečku
- $c_{x,y}$ reprezentuje odsazení příslušné buňky od referenčního bodu fotografie
- $b_{x,y}$ reprezentuje odsazení středu predikovaného ohraničujícího rámečku od referenčního bodu fotografie

Rovnice (20) popisují vztahy, ze kterých je vypočítáno osazení středu ohraničujícího rámečku od referenčního bodu (počátku) fotografie v osách x a y. Rovnice 21 určují výpočet šířky a výšky ohraničujícího rámečku z predikcí a kotev [35]:

$$\begin{aligned} b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{21}$$

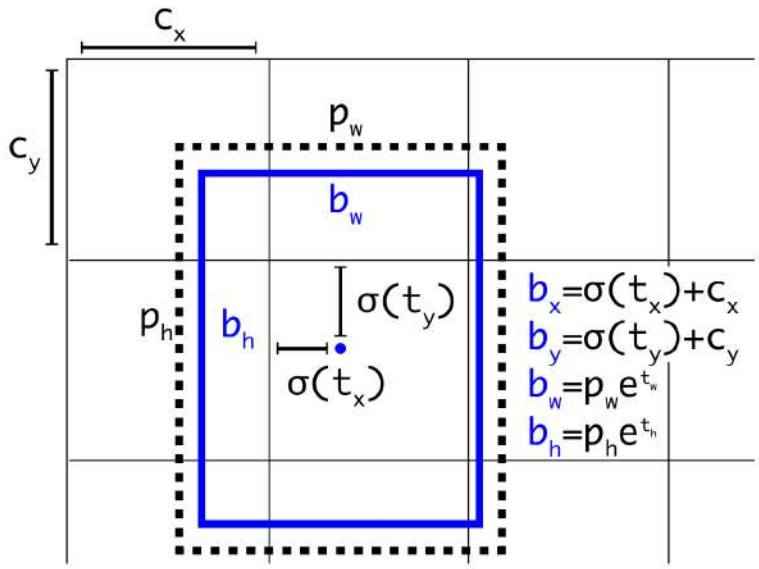
- $t_{w,h}$ reprezentuje výstup sítě výpočet šířky a výšky ohraničujícího rámečku
- $b_{w,h}$ reprezentuje šířku a výšku predikovaného ohraničujícího rámečku

Obrázek 28 znázorňuje počátek souřadnicového systému fotografie v levém horním rohu, střed ohraničujícího rámečku pro detekovaný objekt (modrý rámeček) a výpočet jeho lokalizace i rozměrů z jemu příslušející kotvy (černý tečkovaný rámeček) [35].

Jako kriteriální funkce je při trénování použit součet kvadratické chyby pro výpočet přesnosti umístění a rozměrů ohraničujících rámečků. S tím také přímo souvisí pravděpodobnostní skóre, které je počítáno pro každý rámeček zvlášť. Pro jeho výpočet je použita hodnota *intersection over union* a popisuje ho vztah (22) [34]:

$$p(\text{object}) \cdot IOU(\text{b, object}) = \sigma(t_o) \tag{22}$$

- p(object) pravděpodobnost výskytu objektu
- IOU(b, object) hodnota *intersection over union* předpovězeného a skutečného ohraničujícího rámečku



Obr. 28: Výpočet lokalizace a rozměrů ohraničujícího rámečku [35]

- t_o předpovězená hodnota pro pravděpodobnostní skóre

Výstup sítě pro pravděpodobnost výskytu objektu, podobně jako ostatní výstupy, nabývá hodnot v $\mathbf{R}$. Opět je použita logistická funkce pro přepočítání hodnoty do požadovaného intervalu. Z rovnice (22) $p(\text{object}) \in \{0, 1\}$, buňka střed objektu buď obsahuje nebo naopak. Pokud ho obsahuje, učí se předpovědět pravděpodobnostní skóre jako hodnotu *intersection over union* skutečného a predikovanho rámečku. Tímto síť získá schopnost ohodnotit přesnost předpovězených ohraničujících rámečků. Pro ohraničující rámečky neobsahující žádný střed objektu se nepočítá chyba predikce bounding boxu, ani klasifikace, pouze pravděpodobnostní skóre [35].

5.2.3 Klasifikační třídy

Pro klasifikaci objektu do jedné ze tříd je použita logistická funkce a při trénování binární křížová entropie. Rozdíl proti například první verzi je nepoužití funkce *softmax*, autor zmiňuje, že pro dobrý výkon není potřeba. Tento přístup je výhodný pro datasety jako *Open Images Dataset*, které používají klasifikační třídy jako auto a vozidlo (auto patří zároveň do klasifikační třídy vozidlo). Při použití *softmaxu* je předpokládáno, že každý objekt spadá přesně do jedné třídy, což v detekci objektů není vždy přesné [35].

5.2.4 Predikce v různých škálách

Třetí verze *Yolo* je inspirována sítí *Feature Pyramid Network* z [23], používá predikci ve třech různých rozlišeních. Pro každou buňku jsou predikovány tři ohraničující rámečky. Síť generuje tři tenzory o rozměrech $N \times N \times [3 \cdot (4+1+80)]$. N reprezentuje počet buněk v mřížce (rozlišení), 3 pro počet ohraničujících rámečků predikovaných

jednou buňkou, 4 hodnoty pro určení rozměrů a polohy ohraničujícího rámečku, 1 pro pravděpodobnostní skóre a 80 pro reprezentaci klasifikačních tříd (v datasetu *MS-Coco* je k dispozici 80 tříd) [35].

Na extraktor příznaků je napojeno několik dalších konvolučních vrstev, které jsou součástí pyramidové struktury. Poslední vrstva z této sekvence vypočítá první výstup s nejnižším rozlišením. Dále se pracuje s tenzorem příznaků vypočítaným o dvě konvoluční vrstvy nazpět, ten se převzorkuje na vyšší rozlišení faktorem 2 a bočním spojením je přidán tenzor příznaků vypočítaný dříve v síti. Takto vzniknou příznaky sémanticky bohatší (podrobněji popsáno v kapitole 4.4.4). Tato struktura je analogicky použita i pro druhou a třetí škálu [35].

5.2.5 Extraktor příznaků

Pro extrakci příznaků byla vyvinuta nová síť, která vychází ze starého extraktoru příznaků *Darknet-19*, ale využívá i pokročilejších technologií jako jsou reziduální bloky. Používá konvoluční jádra o velikosti 3×3 a 1×1. Nová verze má dohromady 103 vrstvev v 53 blocích, proto se nazývá *Darknet-53* [35].

Tab. 1: Srovnání extraktorů příznaků [35]

Název sítě	Top-1	Top-5	Počet operací (v miliardách)	FPS
Darknet-19	74,1	91,8	7,29	**171**
ResNet-101	77,1	93,7	19,7	53
ResNet-152	**77,6**	**93,8**	29,4	37
Darknet-53	77,2	**93,8**	18,7	78

Tabulka 1 porovnává různé modely extraktorů příznaků *Darknet* a *ResNet*. Na datasetu *ImageNet* jsou srovnány hodnoty přesnosti, počtu operací a snímků za sekundu. Každý z modelů byl trénován s identickým nastavením a testován na rozlišení 256×256. *Darknet-53* překonává model *ResNet-101*. V oblasti přesnosti je srovnatelný s modelem *ResNet-152*, ale je dvakrát rychlejší [35].

5.2.6 Trénování

Trénování probíhalo na obrázcích z datasetu *MS-Coco*, bylo využito techniky trénování při různých rozlišeních, což umožňuje síti se lépe naučit rozpoznávat objekty. Dále se používají *Bacth Normalization* vrstvy a hodně augmentace dat. Pro trénování i testovaní byl využit framework Darknet [32], což je knihovna pro hluboké učení psaná v jazycích C a CUDA. Framework vytvořil sám autor algoritmu *Yolo* [35].

5.3 Yolo v.4 a další

Čtvrtá verze algoritmu *Yolo* byla představena v roce 2020 v [4]. Společnost Ultralytics [3] pouhé dva měsíce od vydání čtvrté verze publikovala na githubu síť s názvem *Yolo v.5*. K této verzi však neexistuje žádná oficiální vědecká publikace a v komunitě strojového učení se vedou diskuze o relevanci takového pojmenování. Další oficiální verze algoritmu je vylepšení čtvrté verze s názvem *Scaled-Yolo v.4* [44].

6 Praktická část

Tato kapitola se věnuje vlastnímu řešení práce, obsahuje výběr technologií pro vyřešení úlohy, volbu detekčního algoritmu a datasetu, implementaci, trénovací strategie, experimenty s optimalizačními algoritmy a výsledky.

6.1 Technologie

6.1.1 Linux

Pro vývoj byl použit výhradě operační systém *Linux*, konkrétně *Ubuntu 20.04.1* [40]. Tento *open-source* operační systém byl zvolen díky své dostupnosti, spolehlivosti, výkonu a stabilitě. Byl využit jak na straně serveru s grafickou kartou, kde probíhaly výpočty, tak na straně stroje autora, kde probíhal hlavní vývoj.

6.1.2 Python

Python je vysokoúrovňový, interpretovaný, dynamický a objektově orientovaný programovací jazyk. Má jednoduchou syntaxi, velkou standardní knihovnu a správce balíčků *pip* s hlavním repozitářem *PyPI*, odkud je možné stáhnout obrovské množství softwaru třetích strach. *Python* je v současnosti jeden z nejpopulárnějších jazyků pro strojové učení, hluboké učení a umělou inteligenci obecně [31]. Pro výše zmíněné důvody byl pro tento projekt vybrán jazyk *Python*, verze 3.9.12.

6.1.3 Anaconda

Anaconda [1] je distribuce jazyků Python a R pro vědecké výpočty zaměřující se na správu balíčků. *Anacondu* podporují operační systémy *Linux*, *Windows* i *MacOS*. *Conda* je multiplatformní, *open source* správce balíčků a virtuálních prostředí nejen pro *Python*. Pro tento projekt byl zvolen minimalistický instalátor zvaný *Miniconda*. Výhodou je, že všechny balíčky se dají jednoduše exportovat do souboru *enviroment.yml*, ze kterého je možné vytvořit totožné virtuální prostředí. Balíčky použité pro tento projekt jsou dostupné v [30].

6.1.4 PyTorch

PyTorch [27] je *open-source* framework pro strojové a hluboké učení používaná pro počítačové vidění, zpracování přirozeného jazyka. Knihovna je multiplatformní, podporovaná operačními systémy *Linux*, *Windows* i *MacOS*. Knihovna je napsaná v jazycích *Python*, *C++* a *CUDA*, podporuje výpočty na *CPU* i *GPU*. Vývoj financuje společnost *Facebook*, konkrétně *Facebook's AI Research Lab*, která financuje i spoustu výzkumů v oblasti umělé inteligence a hlubokého učení. V tomto projektu byla po-

užita verze *PyTorch 1.11.0* s balíčky pro verzi *CUDA 11.3*. Další použité knihovny spadající pod projekt *PyTorch* jsou:

- **Torchvision** je knihovna pro hluboké učení zaměřená na počítačové vidění. Poskytuje populární datasety, modely sítí, základní tranformace obrazu pro augmentaci a další.
- **Torchmetrics** je knihovna zahrnující přes 80 různých metrik pro hluboké učení, podporuje také tvorbu uživatelských metrik. Výpočty jsou možné na *CPU* i *GPU*.

6.1.5 Tmux

Tmux je terminálový multiplexor napsaný v jazyce C. Umožňuje několika terminálovým relacím zobrazovat se současně v jednom okně a několika programům běžet zároveň. Hlavní funkce *Tmuxu* využitá v projektu je, že umožňuje odpojit proces od terminálu, který jej spustil/řídí. Díky *Tmuxu* bylo možné přes protokol *SSH* spustit trénování na vzdáleném serveru a odpojit se od něj.

6.1.6 Git

Při vývoji sítě byl použit software pro kontrolu verzí *Git*, který usnadnil spoustu práce při implementaci a testování nejrůznějších věcí. Celý projekt byl zálohován na vzdálený server *GitHub* a je volně dostupný v [30].

6.1.7 Hardware

Veškeré výpočty v tomto projektu byly provedeny na serveru *Ústavu automatizace a informatiky* na *Fakultě strojního inženýrství VUT v Brně*. Na serveru je dostupná grafická karta od společnosti *Nvidia*, konkrétně *Nvidia GeForce RTX 3080*.

Alternativa k ústavnímu serveru je služba od společnosti *Google* s názvem *Google Colab*. Tato možnost zůstala nevyužita, díky dostupnosti ústavního serveru nebylo více výpočetní kapacity potřeba.

6.2 Implementace

6.2.1 Dataset

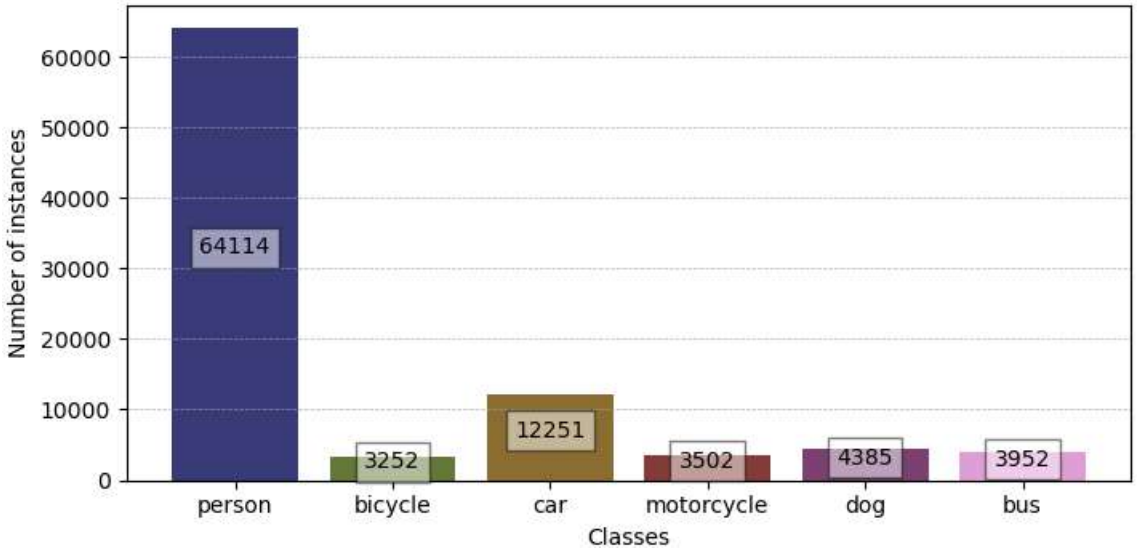
V kapitole 4.6 bylo rozebráno několik dostupných datasetů pro detekci objektů. Z výše uvedených připadal v úvahu dataset *ImageNet*, *PASCAL VOC* a *MS-Coco*. Například dataset *Caltech* by mohl být správnou volbou, pokud by obsahoval také anotace vozidel a né pouze chodců. Konečná volba padla na dataset *MS-Coco*, protože má vyšší průměrnou hustotu instancí objektů na jeden obrázek, což je vhodné pro požadovanou aplikaci. Identifikace účastníků silničního provozu může být, co se týče počtu instancí na jedné fotce, náročný úkol, pokud by takovou identifikaci bylo

potřeba provést například v hustém silničním provozu ve městě. *MS-Coco* je navíc v současnosti jeden z nejpoužívanějších datasetů a stal se *benchmarkem* pro detekční algoritmy.

Jak bylo již výše zmíněno, *MS-Coco* má dohromady 80 klasifikačních tříd. Né všechny jsou k naplnění cílů vhodné (například žirafa, jablko, lžíce). Proto bylo vybráno šest klasifikačních tříd, které byly zhodnoceny jako potenciální účastníci silničního provozu: lidé, psi, kola, auta, autobusy a motorky.

Ze složky bylo odfiltrováno zbylých 74 tříd, taktéž z *json* souboru s anotacemi. Dále byly vymazány nepotřebná data jako například ohraničení objektů pro segmentaci. Tím se velikost anotačního souboru rapidně snížila, a proto se zvýšila rychlost jeho načítání.

V grafech na obrázcích 29, 30 a tabulkách 2, 3 lze vidět rozdělení anotací do klasifikačních tříd. Po odfiltrování jsou třídy v datasetech hodně nevyvážené, početnímu zastoupení jednoznačně dominují lidé, což může působit problémy při trénování.



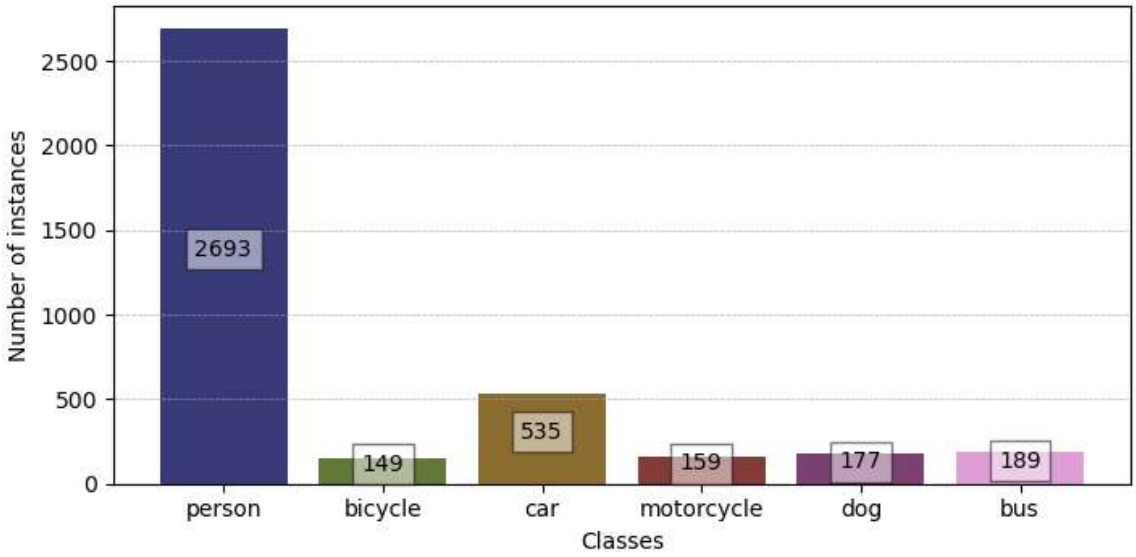
Obr. 29: Nevyvážené zastoupení tříd v trénovacím datasetu

6.2.2 Detekční algoritmus

V kapitole 4 bylo rozebráno několik *single-stage* i *two-stage* detekčních algoritmů a kapitola 5 se pak podrobněji věnuje detekčním sítím z rodiny Yolo. Pro tento projekt byla vybrána síť **Yolo v.3**, která je dobrým kompromisem mezi rychlostí a přesností. Alternativní detekční algoritmy by mohly být například *RetinaNet* nebo *Yolo v.4*, protože jsou oba schopné fungovat v reálném čase, což je pro tuto aplikaci velice vhodné. Velkou roli ve výběru hrála i autorova předešlá znalost algoritmu.

Tab. 2: Zastoupení klasifikačních tříd trénovacího datasetu v počtu anotací

Třída	Počet anotací	Procentuální zastoupení [%]
Lidé	64114	70,1
Psi	4385	4,79
Kola	3252	3,56
Auta	12251	13,4
Autobusy	3952	4,32
Motorky	3502	3,83



Obr. 30: Nevyvážené zastoupení tříd ve validačním datasetu

6.2.3 Struktura projektu

Struktura programu je dělena do čtyř základních složek:

Složka *datasets* obsahuje fotky a anotace použitých datasetů a menší skripty pro filtrování, zobrazování a manipulaci s datasetem.

Adresář *models* obsahuje předtrénovaný model v *pretrained*, natrénované modely pro indentifikaci účastníků silničního provozu a také data o průběhu trénovaní ve složce *train_data*.

Složka *src* obsahuje zdrojový kód k sestavení modelu, nahrání předtrénovaných vah, načtení dat z datasetu, loss funkci, trénovací program, nejrůznější funkce, metriky a mnoho dalších věcí potřebných k natrénování neuronové sítě.

V adresáři *tests* je pár funkcí sloužících pro testování kódu a zobrazování. Z důvodu nedostatku času nebyly řádně naimplementovány testy pro všechny sekce kódu, ale jen pár nezbytných jako test pro správné načtení dat z datasetu a vizualizace predikcí modelu. Ukázka členění složek a souborů v projektu:

Tab. 3: Zastoupení klasifikačních tříd validačního datasetu v počtu anotací

Třída	Počet anotací	Procentuální zastoupení [%]
Lidé	2693	69,02
Psi	177	4,54
Kola	149	3,82
Auta	535	13,71
Autobusy	189	4,84
Motorky	159	4,07

```
project
├── dataset
├── models
│   ├── pretrained
│   │   └── darknet53.conv.74
│   └── train_data
├── src
│   ├── blocks
│   ├── utils
│   ├── config.py
│   ├── dataset.py
│   ├── darknet.py
│   ├── loss.py
│   ├── yolo.py
│   └── yolo_trainer.py
└── tests
    ├── dataset
    │   └── load_images.py
    └── detections
        └── plot_detection.py
```

6.2.4 Architektura

Neuronová síť v projektu byla sestavena po vzoru konfiguračního souboru v [32], protože v originální studii [35] se přesný popis architektury nenachází. Mimo repozitář s kódem sítě [32] publikovaným zároveň se studií [35] algoritmu *Yolo v.3* byl projekt inspirován jednoduchým a přehledným konfiguračním souborem z [29], objektově orientovanou strukturou modelu z [29, 3] založenou na blocích, načítání předtrénovaného modelu po vzoru [3] nebo tzv. *Focal Loss* také z [3].

Z důvodu ušetření času a zdrojů bylo přistoupeno k metodě **trasfer-learning**. Metoda spočívá ve využití předtrénovaného modelu. Tento projekt využil natrénované váhy pro extraktor příznaků z [32].

Při **návrhu modelu** sítě bylo přistoupeno k jejímu rozdělení na dvě části. První částí je extraktor příznaků *Darknet53* a druhá je část, která bude dále nazývána jako *detektor*. Bylo takto učiněno kvůli snadnějšímu načítání předtrénovaných vah a možnosti trénovat každou z těchto částí odděleně. Obě tyto části využívají vícenásobné dědičnosti. První třída, od které dědí je z knihovny *PyTorch* s názvem *torch.nn.Module*, druhá třída zvaná *CNNBuilder*, která byla vytvořena pro účely tohoto projektu. Na základě konfiguračního souboru dokáže sestavit model sítě pro *Darknet53* i *detektor*. Tímto způsobem bylo dosaženo přesunutí metod potřebných k sestavení modelu do zvláštní třídy a kód tříd *Darknetu53* a *detektoru* je čistější. Třída detektoru využívá *konvoluční*, *reziduální* a *predikční* bloky, které jsou společně s třídou *CNNBuilder* součástí balíčku *blocks* ve složce *src/blocks*.

Extraktor příznaků *Darknet-53* má metodu pro **nahrání vah** z předtrénovaného modelu. Originální váhy z [32] jsou k dispozici v binárním souboru a knihovna *PyTorch* přímo nepodporuje jejich nahrávání. Za tímto účelem byla vytvořena třída *WeightsHandler*, která načte soubor a jednotlivým vrstvám přiděluje váhy.

Pro extraktor příznaků jsou v tomto projektu vždy použity váhy z předtrénovaného modelu. Podle [28] je **inicializace vah** velmi důležitou částí při trénování hlubokých neuronových sítí, protože může velkou mírou ovlinit rychlost konvergence tréninku i výslednou kvalitu sítě. Pro *detektor* napojující se na *Darknet-53* je proto využita inicializace vah metodou *Kaiming* pojmenovanou podle autora [15] *Kaiming He*.

Kriteriální funkce je složena ze čtyř hlavních složek:

1. *noobj_loss* je složka počítající chybu v predikci pozadí (buňky bez objektu). Jako kriteriální funkci využívá *binární křížovou entropii*.
2. *obj_loss* je složka, která určuje, jak dobře síť odhadla přesnost predikovaného rámečku. Jako metriku používá *intersection over union* a kriteriální funkci *binární křížovou entropii*.
3. *box_loss* určuje chybu mezi předpovězeným a skutečným ohraničujícím rámečkem. V [35] autor používá *střední kvadratickou chybu*, naopak v [3] je využito metriky *intersection over union*. V rámci tohoto projektu bylo experimentováno s oběma způsoby, nakonec bylo přistoupeno k metodě z [3] kvůli lepším výsledkům z provedených experimentů.
4. *class_loss* používá jako kriteriální funkci *křížovou entropii*. Určuje chybu mezi skutečnými a predikovanými třídami.

Každá z těchto složek je násobena konstantou volitelnou v konfiguračním souboru. Volba těchto konstant silně ovlivňuje vývoj učení sítě. V souboru s kriteriální funkcí byla naimplementována možnost sledovat jednotlivé složky v průběhu tréninku, aby bylo možné lépe zhodnotit průběh učení a případně zasáhnout do tréninku.

Načítání dat z datasetu je naimplementováno prostřednictvím dědičnosti z třídy knihovní *CocoDetection*. Tato třída podporuje nahrávání fotek a anotací datasetu *MS-Coco*. Pomocí magické metody *___getitem___* a pomocné třídy *DataLoader* je možné vytvořit generátor a ve *for cyklu* načítat jednu fotku po druhé nebo několik fotek naráz v *mini-batchi*. Před načtením fotek jsou provedeny transformace, více v kapitole 6.2.5. Z anotací datasetu *MS-Coco* jsou vytvořeny tzv. *target-tezory*. Tyto tenzory mají stejný formát jako jednotlivé výstupní tenzory sítě a jsou určeny k výpočtu chyby predikce.

Pro **trénování** byla vytvořena třída *YoloTrainer*, která spojuje všechny výše popsané části. Je rozdělena na trénovací a testovací smyčku. V trénovací smyčce program iteruje přes trénovací dataset, počítá chybu a pomocí zpětné propagace gradientu upravuje váhy. Testovací smyčka iteruje přes validační dataset, počítá validační chyby a střední průměrnou přesnost (hodnotu mAP). V testovací smyčce je model přepnut do evaluačního módu, ve kterém se vrstvy jako *BatchNormalization* chovají rozdílně. Kvůli náročnosti výpočtů a zrychlení tréninku je validační smyčka aktivovaná jednou za n trénovacích smyček, hodnota n je nastavitelná v konfiguraci.

6.2.5 Augmentace dat

Augmentace dat je běžně používaná technika pro zvětšení velikosti a různorodosti datasetu. V počítačovém vidění se augmentace dat stala běžným nástrojem pro prevenci přeučení sítě. V této práci byla použita augmentace pomocí knihovny *Albumentations* [6].

Za účelem trénování a testovaní byly vztvořeny dva druhy transformací. Testovací transformace přeškáluje obrázek na maximální danou hodnotu, tato práce využívá pouze jedno rozlišení 416×416. Pokud je obrázek šikoroúhlý nebo orientovaný a výšku, je použit *padding*. Stejné transformace jsou provedeny i na trénovací množině, ale navíc jsou přidány: náhodná oříznutí, otočení, změny jasu, kontrastu, nasycení, odstínů, adaptivní vyrovnávání histogramu, přehození kanálů a přepnutí do odstínů šedé.

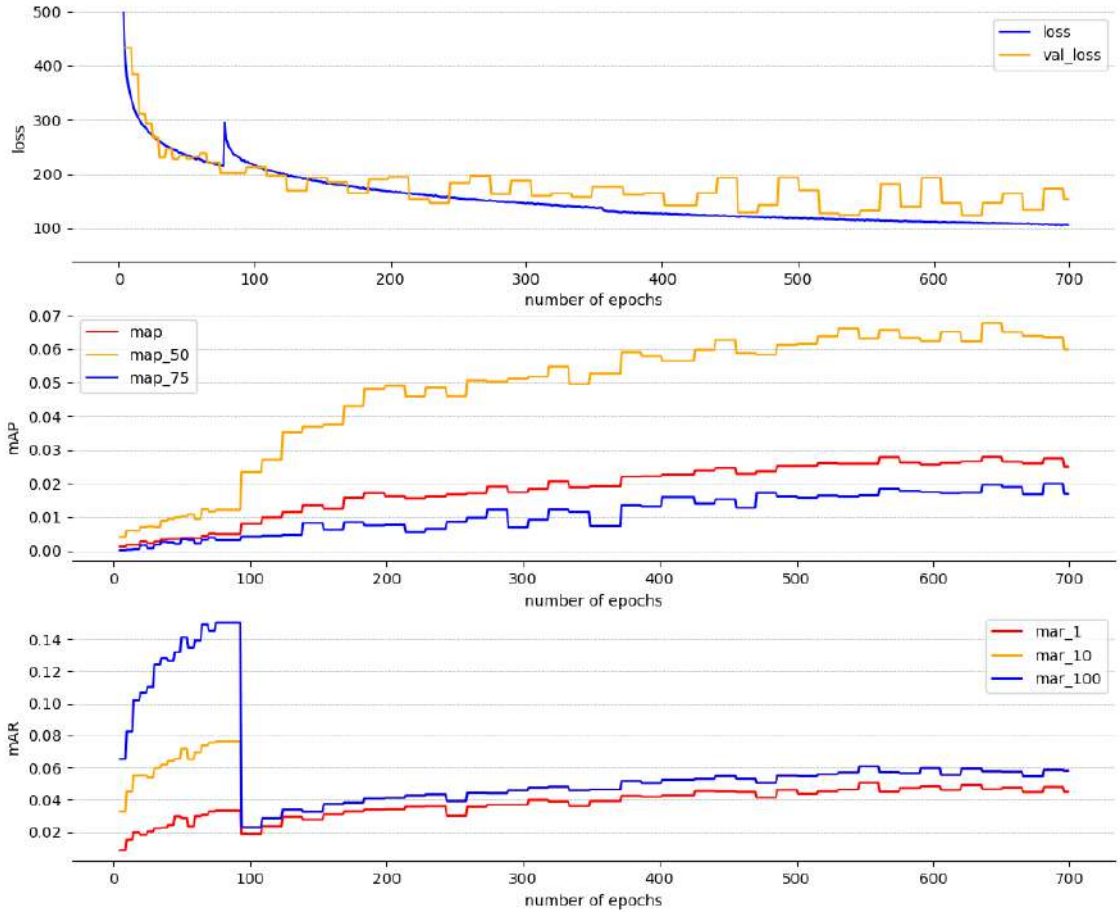
Pro trénovací i testovací fotografie je vždy provedena normalizace. Provedené trasformace se aplikují i na ohraničující rámečky. Knihovna *Albumentations* je kompatibilní s knihovnou *PyTorch*, a proto mohla být využita možnost převést fotografii na datový typ *torch.Tensor*.

6.3 Trénování

Před samotným trénováním probíhalo mnoho testů a experimentů na doladění všech chyb, které se vyskytly až při spojení všech různých částí programu dohromady. K ověření bezchybnosti implementace a schopnosti sítě se učit bylo provedeno trénování i testování na jednom *batchi*. Schopnost sítě přeučit se (tzv. *overfit*) na velmi malé trénovací množině a následná vizualizace detekcí potvrdila správnost naimplementovaných výpočtů.

6.3.1 Trénování se zmraženým extraktorem příznaků

Podle [2] byl v první fázi trénování zmražen extraktor příznaků s načtenými předtrénovanými vahami a učení probíhalo pouze na *detektoru* (druhá část sítě), jehož parametry byly inicializovány metodou *Kaiming*.



Obr. 31: Průběh trénovaní se zmraženým extraktorem příznaků

Grafy na obrázku 31 zobrazují průběh trénování. Prvních 80 epoch bylo trénováno na vyváženější trénovací podmnožině, nikoliv na celém datasetu. *Přesnost* i *recall* v této fázi stoupají. *Recall* (mar_100) se dostal až na hodnotu necelých 15%, přesnost (map_50) však zaostává nad 1%. To znamená, že síť dokáže určitou

část objektů detekovat, spoustu detekcí však neobsahuje žádný objekt (tzv. *false positive*).

Od osmdesáté opochy bylo trénováno na celém datasetu. Na hodnotě trénovací ztrátové funkce lze pozorovat nárůst, hodnota však začne ihned konvergovat. Na hodnotě validační kriteriální funkce žádný nárůst není zobrazen, protože se tato hodnota společně s *přesností* a hodnotou *recall* počítá jednou za n epoch.

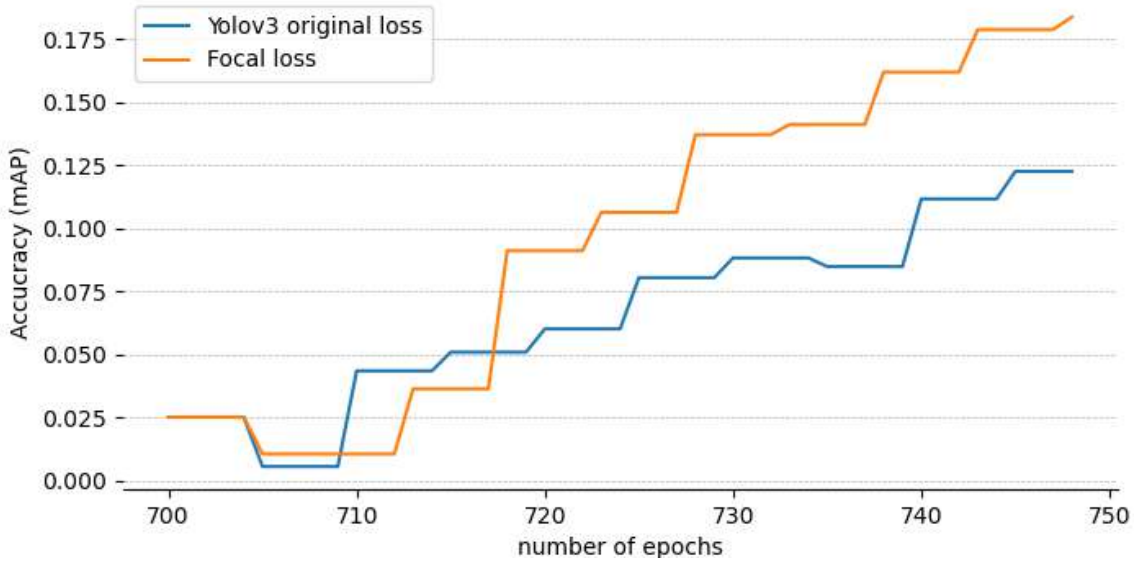
Podle autorů [24] je pro použití kriteriální funkce *focal loss* potřeba pro určité vrstvy sítě speciální inicializace vah, aby bylo možné rozlišit dobře naučenou detekci pozadí, hůře naučitelnou detekci popředí (objektů) a soustředit propagaci gradientu tak, aby více ovlivňovala buňky s horšími výsledky. Trénování s takto nainicializovanými váhami však nemělo v této práci dobrý efekt (ve srovnání s klasickou kriteriální funkcí měla *focal loss* pomalou konvergenci), proto byla v této fázi použita kriteriální funkce po vzoru [35]. Více je o kriteriální funkci psáno v kapitole 6.3.2.

Trénování bylo započato s hodnotou *rychlosti učení (learning rate)* 10^{-5} a velikost batche 64, pro optimalizaci byl využit algoritmus *Adam*. Pro vyšší hodnoty *rychlosti učení* dochází k explozi gradientu a hodnota ztrátové funkce prudce diverguje. Při tréninku byl využit *plánovač rychlosti učení*, který sníží hodnotu *rychlosti učení*, pokud se hodnota validační ztrátové funkce nesnižuje po určitý počet epoch.

Další trénování a experimenty jsou prováděny s celým modelem, jsou při nich tedy upravovány váhy jak extraktoru příznaků, tak detektoru.

6.3.2 Kriteriální funkce

Kriteriální funkce je klíčová pro natrénování sítě. Z hodnoty ztrátové funkce se počítá gradient pro zpětné šíření, při kterém se upravují její parametry. Bylo provedeno několik experimentů se dvěma verzemi ztrátové funkce. Na obrázku 32 je graf jednoho z provedených experimentů srovnávající ztrátovou funkci podle [35] a druhou verzi inspirovanou [3, 24], která využívá metriku *intersection over union* (pro výpočet přesnosti predikce ohraničujícího rámečku) a *focal loss*.

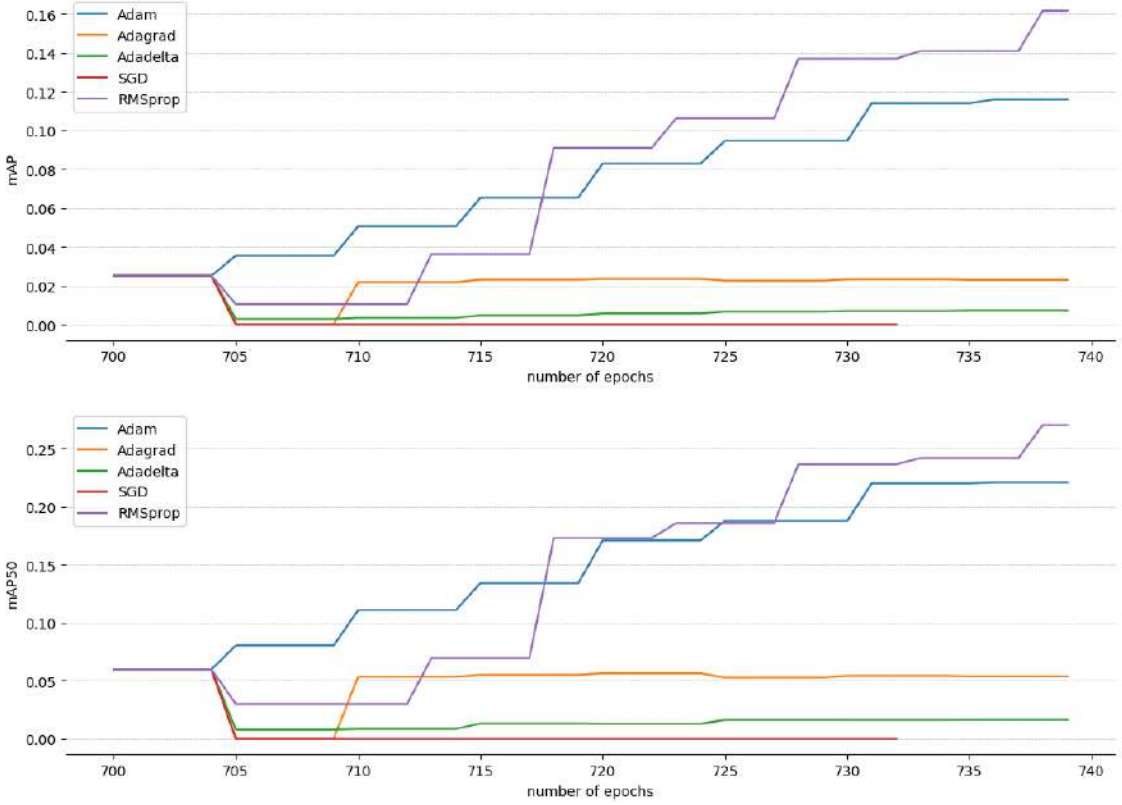


Obr. 32: Srovnání vlivu kriteriálních funkcí na trénování sítě

Hyperparametry sítě byly nastaveny totožně pro obě verze. Byl použit optimalizační algoritmus *RMSprop* (více v kapitole 6.3.3). Parametry pro funkci *focal loss* byly nastaveny podle doporučení autorů [24] $\gamma = 2$, $\alpha = 0,25$ a nebylo s nimi dále nijak experimentováno. Trénovací experiment proběhl na necelých 50 epochách. Síť s originální verzí ztrátové funkce dosáhla přesnosti (mAP) $12,24\%$, modifikovaná verze (mAP) $18,36\%$. Na základě výsledku experimentu byla pro další trénink zvolena modifikovaná verze ztrátové funkce.

6.3.3 Výběr optimalizačního algoritmu

Před dalším trénováním proběhl experiment, ve kterém bylo srovnáno pět optimalizačních algoritmů. Tyto algoritmy byly vybrány z dostupných v knihovně *PyTorch*. Pro každý z algoritmů byly nastaveny totožné hyperparametry. Cílem experimentu bylo zjistit, který z vybraných algoritmů dokáže v daném počtu 40 epoch dosáhnout nejvyšší přesnosti.



Obr. 33: Srovnání optimalizačních algoritmů

Tab. 4: Dosažené hodnoty přesnosti optimalizačních algoritmů

Algoritmus	mAP [%]	mAP-50 [%]	mAP-75 [%]
Adam	11,59	22,09	11,18
Adagrad	2,29	5,38	1,59
Adadelta	0,70	1,60	0,46
SGD	0,00	0,00	0,00
RMSprop	**16,17**	**27,05**	**17,37**

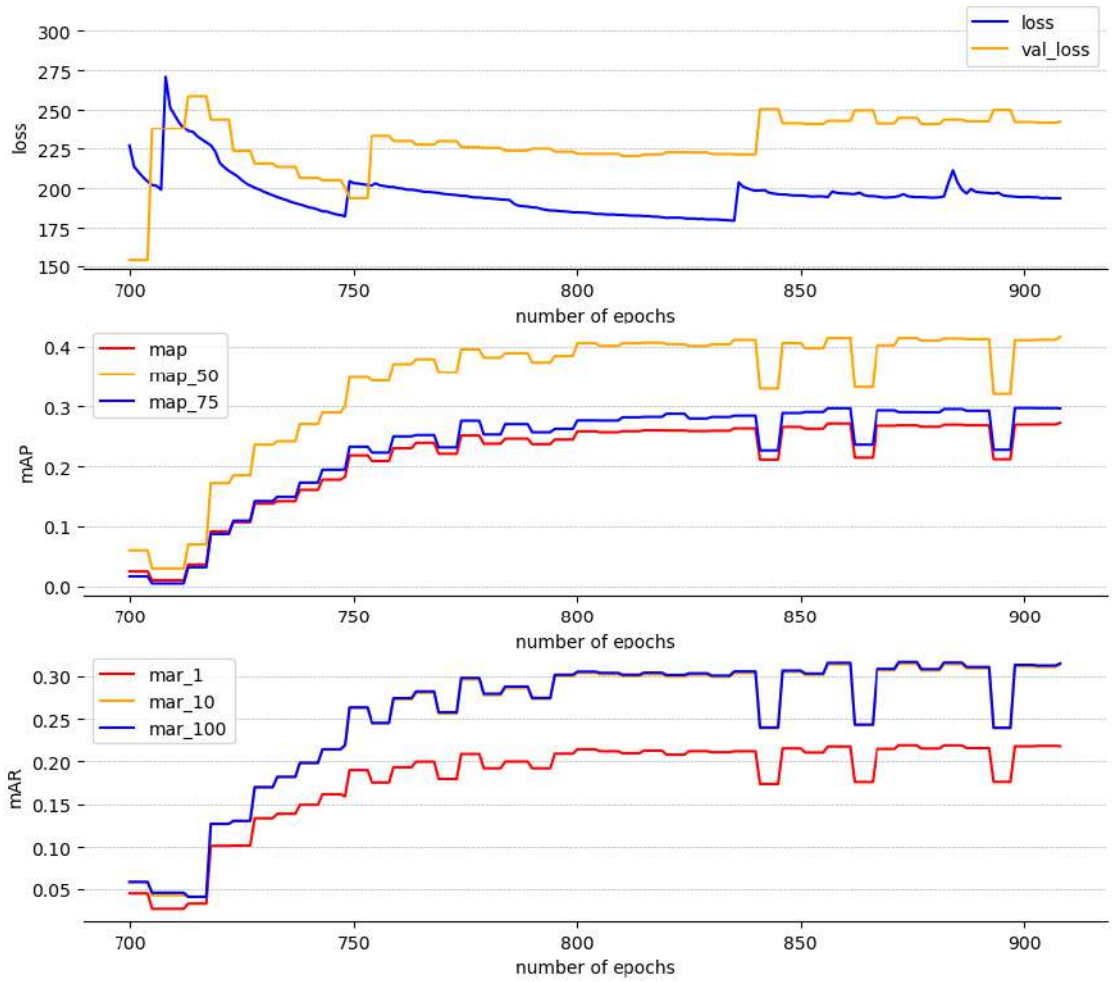
Na obrázku 33 jsou dva grafy srovnávající průběh optimalizace a dosaženou přesnost. Vrchní graf zobrazuje celkovou průměrnou přesnost, spodní graf ukazuje přesnost sítě měřenou mírnější metrikou (mAP50). Algoritmus *SGD* byl z důvodu

ušetření času a zdrojů ukončen po 35. epoše, protože síť jím optimalizovaná dosahovala nulové přesnosti. Konkrétní dosažené hodnoty jsou vypsané v tabulce 4.

Algoritmus *RMSprop* v experimentu dosáhl nejvyšší přesnosti ve všech měřených verzích metriky mAP, a proto byl vyhodnocen jako nejvhodnější pro další trénink se stávajícími hyperparametry sítě. Za špatný výkon algoritmů *Adagrad, Adadelta a SGD* může pravděpodobně konkrétní nastavená konfigurace trénování, hyperparametry sítě a samotné nastavení parametrů těchto optimalizačních algoritmů.

6.3.4 Doladění

Doladění (tzv. *fine-tuning*) je poslední část tréninku. Probíhala s odemčeným extraktorem příznaků. Parametry v jeho vrstvách jsou společně s druhou částí (*detektorem*) součástí procesu optimalizace. Jak bylo výše zmíněno, jako kriteriální funkce byla použita *focal loss* a pro optimalizaci byl využit algoritmus *RMSprop*. Obrázek 34 obsahuje grafy s daty z poslední části učení sítě.



Obr. 34: Průběh poslední části trénování

Pro doladění byla použita síť optimalizovaná algoritmem *RMSprop* v experimentu popsaném v kapitole 6.3.3. V grafu zobrazujícím průběh hodnoty validačních a trénovacích kriteriálních funkcí lze vidět nárůsty v epochách 750, 835 a 885. Ty jsou způsbeny úpravou hodnoty *rychlosti učení* a konstant ovlivňujících velikost jednotlivých složek ztrátové funkce. Při tréninku byly tyto složky sledovány a na základě úsudku autora se modifikovaly konstanty kriteriální funkce.

Z dat na obrázku 34 lze pozorovat mírný pokles validační ztrátové funkce a hodnoty přesnosti i po epoše číslo 800. Tento nárůst však zdaleka není tak rapidní jako před epochou 800. Další trénování a ladění parametrů by pravděpodobně mohlo přinést lepší výsledky, ale z časových důvodů byly výpočty ukončeny.

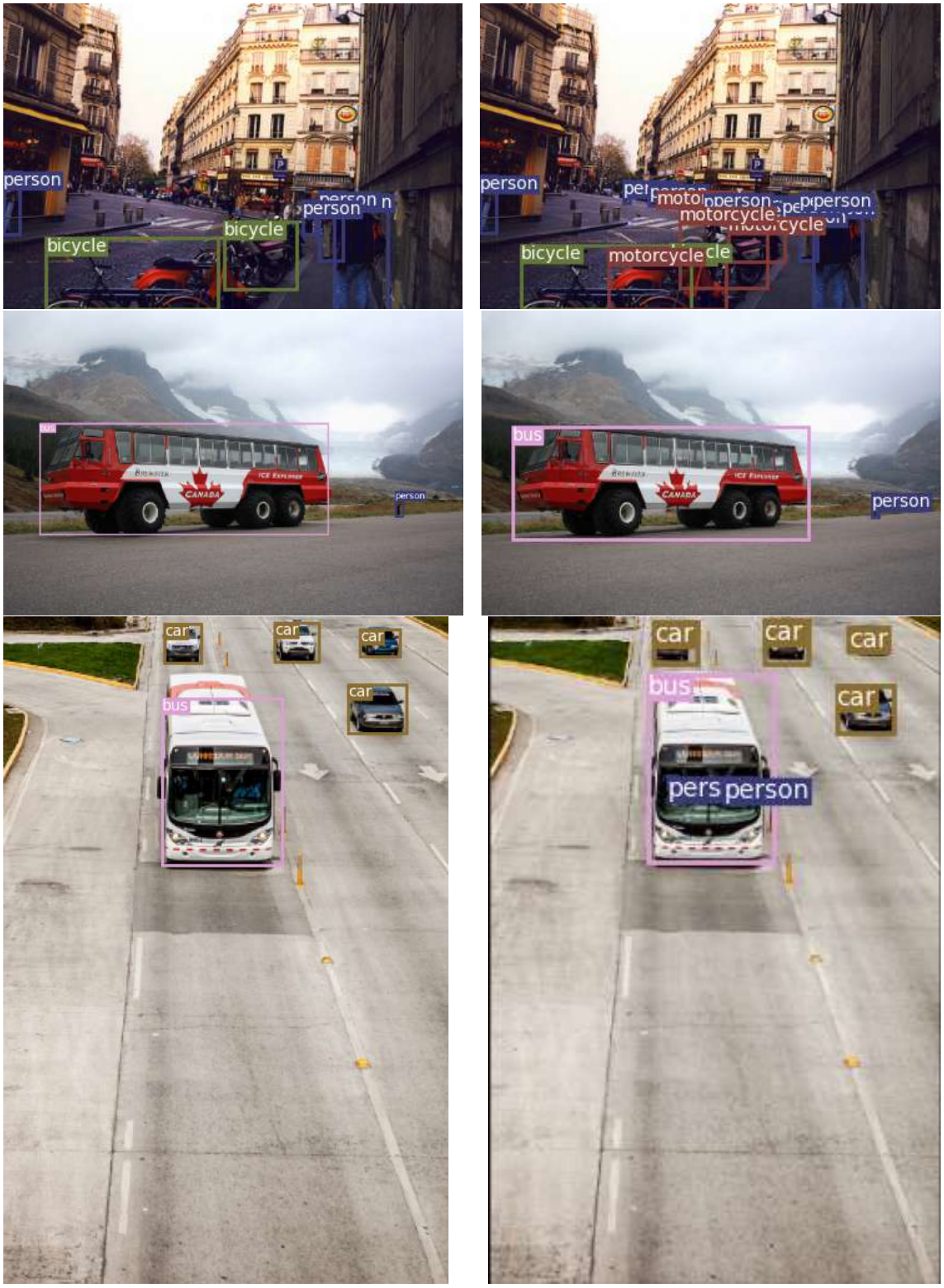
6.4 Výsledky

Síť byla trénována dohromady po 908 epoch. Bylo dosaženo maximální přesnosti (mAP) $27,2\%$ a (mAP-50) $41,67\%$ na výběru z datasetu *MS-Coco*. Nejlépe ze všech klasifikačních tříd umí síť rozpoznávat lidi a auta. To je způsobeno nevyváženým datasetem. Problém byl částečně kompenzován nastavením vah pro jednotlivé u kriteriální funkce *křížové entropie* a *focal loss*, lepší řešení by bylo nadvzorkovat minoritní třídy datasetu pomocí augmentací. V tabulce 5 lze vidět srovnání výsledků této práce se sítěmi z [35]. Obrázky 35 a 36 porovnávají detekce sítě s anotacemi z datasetu.

Tab. 5: Srovnání výsledků práce s originálním algoritmem [35]

Síť	**tato práce**	Yolov3	Yolov3
rozlišení	416×416	416×416	608×608
mAP [%]	27,25	31,0	33,0
mAP-50 [%]	41,67	-	57,9
mAP-75 [%]	29,61	-	34,4

Obr. 35: Srovnání detekcí a anotací. Vlevo je detekce sítě, vpravo naanotovaná fotka [26]



Obr. 36: Srovnání detekcí a anotací. Vlevo je detekce sítě, vpravo naanotovaná fotka [26]

7 ZÁVĚR

V praktické části byly popsány použité technologie, datasety, architektura, provedené experimenty, zvolená učící strategie, optimalizační funkce a výsledky. Na datasetu *MS-Coco* bylo dosaženo maximální přenosti (mAP) $27,25\%$ pro detekci účastníků silničního provozu. Studie, ve které byl algoritmus publikován [35] uvádí přesnost (mAP) $31,0\%$ na stejném datasetu.

Autor ve [35] trénoval svůj model na více grafických kartách a trénink sítě byl uskutečněn pro více rozlišení. Tyto dvě věci nebyly v této práci provedeny a jsou dobrým námětem pro zlepšení stávajícího modelu. Další zlepšení by mohlo být provedeno vyvážením použitého datasetu jeho nadvzorkováním pomocí augmentace dat.

Pro natrénování modelu byla použita metoda *transfer-learning*. Jako kriteriální funkce se osvědčila *focal loss*. Model byl optimalizován algoritmy *Adam* a *RMSprop*.

Síť vytvořena v této práci dokáže detekovat objekty vyskytující se v silničním provozu. Největší problém ji činí detekce menších objektů v hustých skupinách a přesné zarovnání rámečků s objektem.

8 SEZNAM POUŽITÉ LITERATURY

[1] *Anaconda Software Distribution*. 2020.
URL `https://docs.anaconda.com/`

[2] Abadi, M.; Agarwal, A.; Barham, P.; aj.: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015, software available from tensorflow.org.
URL `https://www.tensorflow.org/`

[3] et. al., G. J.: *ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support*. `https://doi.org/10.5281/zenodo.5563715`, oct 2021.

[4] Bochkovskiy, A.; Wang, C.-Y.; Liao, H.-Y. M.: *Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934*, 2020.

[5] Brownlee, J.: *Deep learning for computer vision: image classification, object detection, and face recognition in python*. Machine Learning Mastery, 2019.

[6] Buslaev, A.; Iglovikov, V. I.; Khvedchenya, E.; aj.: *Albumentations: fast and flexible image augmentations. Information*, ročník 11, č. 2, 2020: str. 125.

[7] Deng, J.; Xuan, X.; Wang, W.; aj.: *A review of research on object detection based on deep learning*. In *Journal of Physics: Conference Series*, ročník 1684, IOP Publishing, 2020, str. 012028.

[8] Esteva, A.; Chou, K.; Yeung, S.; aj.: *Deep learning-enabled medical computer vision. NPJ digital medicine*, ročník 4, č. 1, 2021: s. 1–9.

[9] Forsyth, D. A.; Ponce, J.: *A modern approach. Computer vision: a modern approach*, ročník 17, 2003: s. 21–48.

[10] Gauen, K.; Dailey, R.; Laiman, J.; aj.: *Comparison of visual datasets for machine learning*. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, IEEE, 2017, s. 346–355.

[11] Ge, Z.; Liu, S.; Wang, F.; aj.: *Yolox: Exceeding yolo series in 2021. arXiv preprint arXiv:2107.08430*, 2021.

[12] Girshick, R.: *Fast r-cnn*. In *Proceedings of the IEEE international conference on computer vision*, 2015, s. 1440–1448.

[13] Girshick, R.; Donahue, J.; Darrell, T.; aj.: *Rich feature hierarchies for accurate object detection and semantic segmentation.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, s. 580–587.

[14] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep learning.* MIT press, 2016.

[15] He, K.; Zhang, X.; Ren, S.; aj.: *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.* In *Proceedings of the IEEE international conference on computer vision*, 2015, s. 1026–1034.

[16] He, K.; Zhang, X.; Ren, S.; aj.: *Deep residual learning for image recognition.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, s. 770–778.

[17] Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; aj.: *Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580*, 2012.

[18] Ioffe, S.; Szegedy, C.: *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In *International conference on machine learning*, PMLR, 2015, s. 448–456.

[19] Kaggle: *Diabetic Retinopathy Detection.* [online]: https://www.kaggle.com/c/diabetic-retinopathy-detection, 2022.

[20] Kathuria, A.: *What's new in yolo v3. Towards data science*, ročník 23, 2018.

[21] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: *Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems*, ročník 25, 2012.

[22] LeCun, Y.; Bengio, Y.; Hinton, G.; aj.: *Deep learning. nature, 521 (7553), 436-444. Google Scholar Google Scholar Cross Ref Cross Ref*, 2015.

[23] Lin, T.-Y.; Dollár, P.; Girshick, R.; aj.: *Feature pyramid networks for object detection.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, s. 2117–2125.

[24] Lin, T.-Y.; Goyal, P.; Girshick, R.; aj.: *Focal loss for dense object detection.* In *Proceedings of the IEEE international conference on computer vision*, 2017, s. 2980–2988.

[25] Lin, T.-Y.; Maire, M.; Belongie, S.; aj.: *Microsoft coco: Common objects in context.* In *European conference on computer vision*, Springer, 2014, s. 740–755.

[26] Lin, T.-Y.; Maire, M.; Belongie, S.; aj.: *Microsoft coco: Common objects in context*. In *European conference on computer vision*, Springer, 2014, s. 740–755.

[27] Paszke, A.; Gross, S.; Chintala, S.; aj.: *Automatic differentiation in PyTorch*. 2017.

[28] Patel, P.; Nandu, M.; Raut, P.: *Initialization of weights in neural networks. Int. J. Sci. Eng. Dev. Res*, ročník 3, č. 11, 2018: s. 73–79.

[29] Persson, A.: *Machine-Learning-Collection*. `https://github.com/aladdinpersson/Machine-Learning-Collection`, 2022.

[30] Provazník, M.: *Yolov3*. 5 2022, doi:10.5281/zenodo.1234.
URL `https://github.com/AddictionLord/Yolov3`

[31] Raschka, S.; Mirjalili, V.: *Python machine learning: Machine learning and deep learning with python. Scikit-Learn, and TensorFlow. Second edition ed*, ročník 10, 2017: str. 3175783.

[32] Redmon, J.: *Darknet: Open Source Neural Networks in C*. `http://pjreddie.com/darknet/`, 2013–2016.

[33] Redmon, J.; Divvala, S.; Girshick, R.; aj.: *You only look once: Unified, real-time object detection*. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, s. 779–788.

[34] Redmon, J.; Farhadi, A.: *YOLO9000: better, faster, stronger*. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, s. 7263–7271.

[35] Redmon, J.; Farhadi, A.: *Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767*, 2018.

[36] Ren, S.; He, K.; Girshick, R.; aj.: *Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems*, ročník 28, 2015.

[37] Russakovsky, O.; Deng, J.; Su, H.; aj.: *Imagenet large scale visual recognition challenge. International journal of computer vision*, ročník 115, č. 3, 2015: s. 211–252.

[38] Sanezoo: *Universal visual inspection. Flexible bin picking.* [online]: https://www.sanezoo.com/, 2022.

[39] Snavely, N.; Seitz, S. M.; Szeliski, R.: *Photo tourism: exploring photo collections in 3D*. In *ACM siggraph 2006 papers*, Association for Computing Machinery, 2006, s. 835–846.

[40] Sobell, M. G.: *A practical guide to Ubuntu Linux*. Pearson Education, 2015.

[41] 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)*, IEEE, 2018, s. 209–214.

[42] Szeliski, R.: *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[43] Thuan, D.: *Evolution of yolo algorithm and yolov5: the state-of-the-art object detection algorithm*. 2021.

[44] Wang, C.-Y.; Bochkovskiy, A.; Liao, H.-Y. M.: *Scaled-yolov4: Scaling cross stage partial network*. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, 2021, s. 13029–13038.

[45] X-Sight: [online]: https://www.xsight.eu/, 2022.

[46] Zaidi, S. S. A.; Ansari, M. S.; Aslam, A.; aj.: *A survey of modern deep learning based object detection models. Digital Signal Processing*, 2022: str. 103514.

[47] Zou, Z.; Shi, Z.; Guo, Y.; aj.: *Object detection in 20 years: A survey. arXiv preprint arXiv:1905.05055*, 2019.

9 Přílohy

Zdrojový kód [30] `https://github.com/AddictionLord/Yolov3`