VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

MODEL HLUBOKÉHO UČENÍ VHODNÝ PRO VIZUÁLNÍ DETEKCI A KLASIFIKACI OBECNÉHO OBJEKTU Z PRŮMYSLU

DEEP LEARNING MODEL FOR VISUAL DETECTION AND CLASSIFICATION GENERAL OBJECT FROM INDUSTRY

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE Radim Dočkal

AUTHOR

VEDOUCÍ PRÁCE Ing. Lukáš Kratochvíla

SUPERVISOR

BRNO 2021

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Radim Dočkal ***ID:*** 211140
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Model hlubokého učení vhodný pro vizuální detekci a klasifikaci obecného objektu z průmyslu

POKYNY PRO VYPRACOVÁNÍ:

V průmyslu se setkáváme s objekty různých tvarů a materiálu. Hluboké učení nabízí prostředky pro detekci a klasifikaci i těžko popsatelných výrobků. Cílem práce je vytvořit model hlubokého učení schopný vizuální detekce a klasifikace obecného objektu (např. šroub či sušenka).

1. Vytvoření rešerše na state-of-the-art architektur, implementací a modelů hlubokého učení.
2. Výběr vhodného modelu pro klasifikaci a detekci obecného objektu.
3. Sestavení nebo výběr vhodného datasetu pro doučení a hodnocení modelu.
4. Provedení doučení či úprava modelu.
5. Vyhodnocení vytvořeného modelu na testovacím datasetu.

DOPORUČENÁ LITERATURA:

[1.] BENGIO, Yoshua; GOODFELLOW, Ian; COURVILLE, Aaron. Deep learning. Massachusetts, USA:: MIT press, 2017.

[2.]NIELSEN, Michael A. Neural networks and deep learning. San Francisco, CA: Determination press, 2015.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Lukáš Kratochvíla

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Cílem této bakalářské práce je naprogramovat model hlubokého učení pro vizuální detekci a klasifikaci obecného objektu z průmyslu. Práce je rozdělena do pěti kapitol. První kapitola se zabývá rešerší nejpoužívanějších architektur tohoto typu. Druhá kapitola se zabývá výběrem nejvhodnější architektury pro použití v průmyslu. Ve třetí kapitole je popsán postup vytváření vlastního datasetu. Ve čtvrté kapitole je pak popsán celý proces samotné implementace modelu tak, aby každá dílčí část architektury byla dostatečně vysvětlena a v páté kapitole jsou popsány výsledky. Shrnutí výsledků a doporučené procedury pro případnou implementaci v reálném prostředí jsou k nalezení v závěru této práce.

Klíčová slova

počítačové vidění, hluboké učení, model hlubokého učení, detekce a klasifikace objektu, detekce objektu, YOLOv3, YOLO, You Only Look Once, Python, PyTorch

Abstract

The goal of this bachelor's thesis is to programme deep learning model for visual detection and classification of general object from industry. The paper is divided into five chapters. First chapter deals with research of the most used architectures of this type. The second chapter deals with choosing the best fitting architecture for usage in industry. In the third chapter is desribed the procedure of creating your own dataset. The fourth chapter then describes the implementation process itself, so that each sub-part of the architecture was sufficiently described and the results are described in the fifht chapter. The summary and recommended procedures for potential implementation in real environment can be found in the conclusion of this paper.

Keywords

computer vision, deep learning, deep learning model, object detection and classification, object detection, YOLOv3, YOLO, You Only Look Once, Python, PyTorch

Bibliografická citace

DOČKAL, Radim. *Model hlubokého učení vhodný pro vizuální detekci a klasifikaci obecného objektu z průmyslu.* Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/133736. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Lukáš Kratochvíla.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Radim Dočkal*
VUT ID studenta:	*211140*
Typ práce:	*Bakalářská práce*
Akademický rok:	*2020/21*
Téma závěrečné práce:	*Model hlubokého učení vhodný pro detekci a klasifikaci obecného objektu z průmyslu*

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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 vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.
V Brně dne: 21. května 2021 --------------------------------------

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Lukáši Kratochvílovi za vstřícnou pomoc a cenné rady, které mi byly poskytnuty při zpracování mé bakalářské práce.

V Brně dne: 21.května 2021 --------------------------------------

podpis autora

Obsah

SEZNAM OBRÁZKŮ **8**

SEZNAM TABULEK **9**

ÚVOD **10**

1. STATE-OF-ART ARCHITEKTURY **12**

1.1 R-CNN 12
1.2 YOLO 14
1.3 SSD A RETINANET 16

2. VÝBĚR VHODNÉHO MODELU **18**

2.1 RYCHLOST 18
2.2 PŘESNOST 20
2.3 ROZHODNUTÍ 21

3. DATASET **22**

3.1 PŘÍPRAVY 22
3.1.1 Nastavení scény *22*
3.1.2 Klasifikační třídy *23*
3.2 ÚPRAVA FOTOGRAFIÍ 23
3.3 ANOTACE A AUGMENTACE 24
3.4 POPIS VÝSLEDNÉHO DATASETU 25

4. IMPLEMENTACE **27**

4.1 YOLOV3 27
4.2 DATASET 29
4.3 TARGET CONVERSION 30
4.4 INTERSECTION OVER UNION 31
4.5 NON-MAX SUPPRESSION 32
4.6 MEAN AVERAGE PRECISION 33
4.7 LOSS FUNKCE 33
4.8 CONFIG, TRAIN A SHOW IMAGE 34

5. UČENÍ A TESTOVÁNÍ **37**

5.1 UČENÍ 37
5.2 TESTOVÁNÍ 37

6. ZÁVĚR **41**

LITERATURA **42**

SEZNAM PŘÍLOH **46**

SEZNAM OBRÁZKŮ

1.1 Princip R-CNN architektury [10] 12
1.2 Algoritmus selektivního hledání [11] 12
1.3 Porovnání rychlostí sítí R-CNN, SPP-Net, Fast R-CNN a Faster R-CNN [13] 13
1.4 Architektura Faster R-CNN [14] 13
1.5 Příklad masky vytvořené architekturou Mask R-CNN [17] 14
1.6 Obraz v architektuře YOLOv1 rozdělen do mřížky 7x7 [19] 15
1.7 Ukázka funkce anchorů [21] 15
1.8 Ačkoliv se principem fungování může zdát, že architektury SSD a YOLO jsou stejné, není tomu tak [21] 16
1.9 Architektura RetinaNet [24] 17
2.1 Výsledky publikované v článku RetinaNet na datasetu COCO [24] 19
2.2 Výsledky publikované v článku YOLOv3 na datasetu COCO [22] 19
2.3 Výsledky publikované v článku Light-Weight RetinaNet na datasetu COCO [28] 19
3.1 Náčrt fotografovací scény 23
3.2 Ukázka fotografie připínáčku s rozlišením 600x400 px 24
3.3 Příklad anotace a augmentace na fotografii kancelářské sponky 26
4.1 Demonstrační model architektury YOLOv3, kde jsou důležité 3 výstupy. Výstup 1 (scale 1) je rozdělen do mřížky s buňkami o velikosti 32x32 px, výstup 2 (scale 2) obsahuje buňky o velikosti 16x16 px a poslední výstup (scale 3) obsahuje buňky o velikosti 8x8 px [33] 27
4.2 Darknet-53 [22] 28
4.3 Vývojový diagram úpravy obrázků a 2 ukázky výstupních obrázků 29
4.4 Transformace z tvaru bounding boxu (hnědá, červená) na tvar targetů (bílá, bledě modrá. Nenulové hodnoty jsou zde jen orientační a neodpovídají skutečným spočteným hodnotám. 31
4.5 Vývojový diagram výpočtu IoU, kde x a y vstupních bounding boxů vyjadřuje pozici středu objektu 31
4.6 Non-max suppression [39] 32
4.7 Vývojový diagram procesu NMS 32
4.8 Predikce po krátkém testovacím učení. Bílý rámeček značí ground-truth, ostatní bounding boxy jsou predikce sítě. Confidence predikovaných bounding boxů jsou hodnoty před použitím funkce sigmoid. 36
5.1 Loss a mAP hodnoty v průběhu celého učení. Červená – trénovací loss, modrá – validační loss, zelená – validační mAP 37
5.2 Predikce architektury YOLOv3 39
5.3 IoU 90 % 40

SEZNAM TABULEK

2.1 Výsledky publikované v článku YOLOv2 (YOLO9000) na datasetu PACAL VOC2007 [20] 18
5.1 Výsledky implementace YOLOv3. Model A značí instanci, která v průběhu učení dosáhla nejnižší hodnoty chyby na validačním datasetu, model B označuje instanci, která v průběhu učení dosáhla nejvyšší hodnoty mAP na validačním datasetu 38

ÚVOD

Umělá neuronová síť je počítačový program, který je naprogramován tak, aby svou architekturou dokázal řešit úlohy za pomocí učení z vlastních chyb. Je složena z umělých neuronů, které jsou navzájem propojeny, a tím svou analogií připomínají biologické neurony. Zatímco v lidském mozku neurony fungují tak, že buď přenáší, nebo nepřenáší vzruch [1], což v podstatě odpovídá binární logice, umělé neurony mohou mít na výstupu jakoukoliv hodnotu, nejčastěji od -1 do 1. Nynější umělé neuronové sítě nejčastěji obsahují několik desítek až stovek milionů umělých neuronů [2]. Pro srovnání, lidský mozek obsahuje několik desítek miliard neuronů [3].

Umělé neurony se skládají z tzv. „vah" a „biasů" a obvykle se skládají do vrstev. Nejčastěji se používají tři, nebo čtyři vrstvy, přičemž první vrstva funguje jako vstupní a poslední vrstva jako výstupní. Pokud existuje síť takovýchto neuronů, pak je vhodné pro výpočty využít maticové operace. Protože umělé neurony pracují s číselnými hodnotami, musíme na vstup vkládat data v číselné formě. Pokud se tedy na vstup vkládá obraz, video, text apod., musí se tato data předzpracovat do tenzorů čísel.

Na základě vstupních dat a hodnot vah a biasů se pak přes maticové operace na výstupu objeví žádaný výsledek. To ovšem platí až v případě, že neuronová síť je tzv. „naučená". Při první inicializaci se ve vahách a biasech objeví náhodné hodnoty, které musí být nejprve podrobeny učení. Nejčastější typ učení je učení s učitelem (supervised learning). Ten funguje tak, že se na vstup vloží data, která mají být zpracována a výsledek, který neuronová síť spočítá se pak porovná se správným výsledkem, a poté se na základě tzv. algoritmu zpětného učení (back-propagation algorithm), který využívá gradienty, upraví hodnoty vah a biasů v síti tak, aby byl příště výsledek blíže správnému řešení. Tento krok se opakuje tak dlouho, dokud neuronová síť nemá uspokojující výsledky.

Je důležité zmínit, že pro správné učení sítí je nutné disponovat velkými soubory dat, díky kterým je proces učení vůbec možný. Pro ty nejběžnější problémy dnes naštěstí existuje spousta datasetů. Pokud ale má síť řešit méně běžnou problematiku, nebo pokud je na dataset vyžadován nějaký požadavek, pak s nejvyšší pravděpodobností bude nutné dataset vytvořit.

V dnešní době existuje obrovské množství různých architektur, kde každá je vhodná pro jinou problematiku, a nejen díky tomu se umělé neuronové sítě dnes využívají ve velkém množství různých průmyslových oblastí. Marketingové společnosti je využívají k cíleným reklamám, ve zdravotnictví pomáhají s diagnostikami, v herním průmyslu se využívají k autonomnímu řízení, v lingvistice neuronové sítě pomáhají s překlady, nebo porozuměním textu a vědcům pomáhají k vyřešení problémů, které jsou těžko vyřešitelné klasickými způsoby, jako například zkoumání vesmíru [4][5][6][7][8].

Při práci s obrazy se neuronové sítě nejčastěji využívají ke dvěma druhům úloh. První druh je obyčejná klasifikace, kde se na vstup neuronové sítě vloží obraz a síť na výstupu

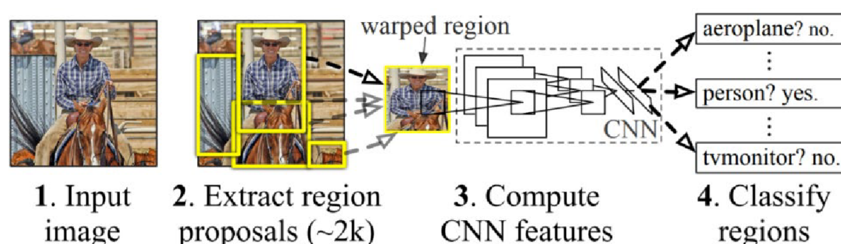
určí, co je na obraze. Tento typ úloh je relativně jednoduchý, protože při učení se pouze porovnává, jestli síť správně klasifikovala objekt v obraze. Druhá oblast se zabývá nejen klasifikací, ale i detekcí objektu. To znamená, že síť musí nejen správně určit jaký objekt v obraze je, ale i přesně určit na kterém místě v obraze se nachází. Tato neuronová síť navíc dokáže detekovat a klasifikovat i více objektů v jednom obraze. Učící proces i samotná síť je pak složitější, protože kromě obyčejné klasifikace objektů musí neuronová síť určovat i pozice objektů v obraze.

1. STATE-OF-ART ARCHITEKTURY

Nejjednodušší a nejpřirozenější přístup k detekci objektu v obraze je takový, že by se z každé fotografie vždy vyřízly nějaké části, a ty by byly následně analyzovány architekturou pro klasifikaci objektu v obraze. Jak ale bude popsáno dále v této kapitole, tento přístup je velmi neefektivní, a proto zde budou popsány principy nejpoužívanějších architektur pro detekci objektu v obraze. Dnes nejvyužívanějšími architekturami pro tuto problematiku jsou architektury R-CNN, SSD, RetinaNet a YOLO[9].

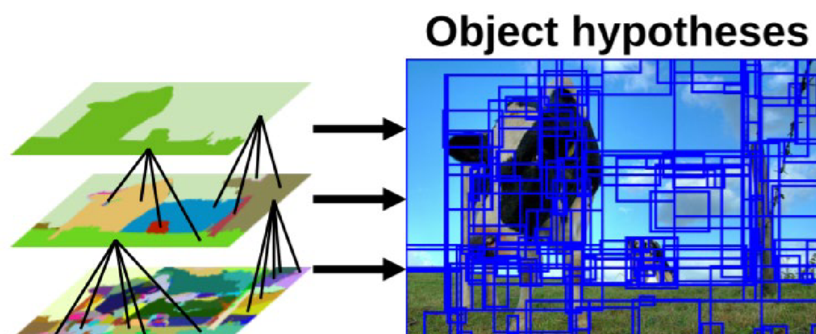
1.1 R-CNN

R-CNN architektura pro detekci objektů v obraze je nejstarší z těch, které zde budou uvedeny. Byla vytvořena v roce 2014 [10]. Od té doby vzniklo ještě nespočet různých modifikací této architektury, mezi něž například patří Fast R-CNN, Faster R-CNN, nebo například Mask R-CNN. Právě díky těmto modifikacím je tento typ architektury dodnes jedním z nejpoužívanějších.



Obrázek 1.1 Princip R-CNN architektury [10]

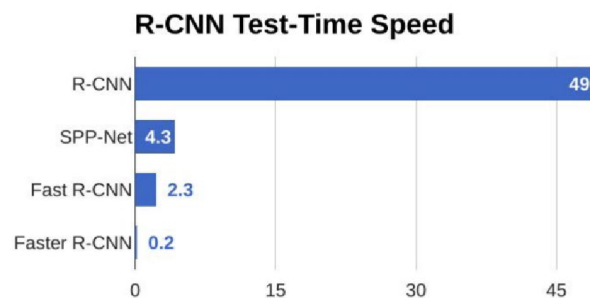
Z obrázku 1.1 je vidět, jak architektura R-CNN funguje. Na vstup sítě je vložen obrázek. Ten je předzpracován algoritmem pro selektivní hledání, který obrázek zpracuje tak, že na výstupu je přibližně 2000 „navrhnutých“ oblastí. Každá tato oblast je poté vložena na vstup konvoluční neuronové sítě (CNN), která pak má za úkol zjistit, jestli



Obrázek 1.2 Algoritmus selektivního hledání [11]

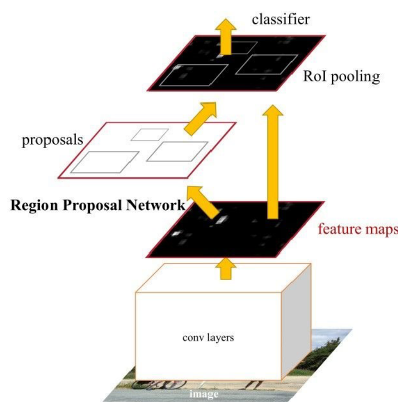
je v oblasti nějaký konkrétní objekt a následně určit, o jaký objekt se jedná. Algoritmus pro selektivní hledání funguje tak, že se obraz rozdělí do několika segmentů např. podle barvy, kontrastu apod. [11]. Na obr. 1.2 této prvotní segmentaci odpovídá levý dolní obrázek. Poté se postupně pospojují segmenty s velmi podobnými vlastnostmi. Výsledné segmenty jsou následně označeny jako navrhnuté oblasti.

V roce 2015 přišlo vylepšení této architektury s názvem Fast R-CNN [12]. Zatímco v R-CNN architektuře byla každá navrhnutá oblast samostatně vložena do konvoluční neuronové sítě [10], ve Fast R-CNN architektuře jsou tyto oblasti do konvoluční neuronové sítě vkládány všechny najednou [12]. Na výstupu této sítě pak je tzv. „feature map“, která je poté ještě vložena do plně propojené sítě („fully connected network“). Na výstupu této vrstvy je pak výsledek, který určuje objekty v obraze [12].



Obrázek 1.3 Porovnání rychlostí sítí R-CNN, SPP-Net, Fast R-CNN a Faster R-CNN [13]

Na obrázku 1.3 lze vidět, že architektuře R-CNN trvá vyhodnotit jeden testovací obrázek necelých 50 sekund, což je velmi pomalé. Naopak architektuře Fast R-CNN trvá vyhodnocení jednoho obrázku něco málo přes 2 sekundy. Tato hodnota je mnohem lepší, ale lze si všimnout, že architektuře Faster R-CNN trvá vyhodnocení jednoho obrázku dokonce jen 0,2 sekundy. To je další téměř desetinásobné zrychlení. I přesto existují mnohem rychlejší architektury, které jsou popsány v dalších kapitolách.



Obrázek 1.4 Architektura Faster R-CNN [14]

Architektura Faster R-CNN je další modifikace architektury R-CNN [15]. Na obr. 1.4 je vidět, jak tato architektura principiálně funguje. Faster R-CNN nevyužívá algoritmus selektivního hledání a místo toho je zde konvoluční síť, která rychleji určí navrhované oblasti [15]. Tyto oblasti jsou pak společně s feature map zpracovány, a tím vzniknou predikované bounding boxy [15]. Architektura Faster R-CNN také přišla s tzv. „anchory“, které jsou více popsány v sekci 1.2 [15].

Další modifikací sítě R-CNN je Mask R-CNN, která vznikla v roce 2017 [16]. Ta je velmi podobná Faster R-CNN architektuře, která má na výstupu pouze typ objektu v obraze a souřadnice objektu. Mask R-CNN navíc obsahuje výstup, který určuje masku objektu v obraze (viz. obr. 1.5) [16]. Tato síť dokáže zpracovat 5 snímků za sekundu, což odpovídá času 0,2 sekundy pro zpracování jednoho obrazu [16].

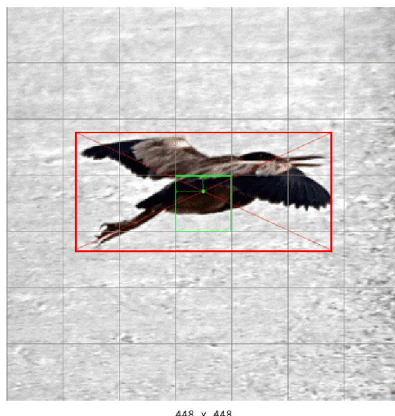
Obrázek 1.5 Příklad masky vytvořené architekturou Mask R-CNN [17]

1.2 YOLO

Architektura YOLOv1 vznikla v roce 2016 [18]. Tato architektura jde odlišnou cestou, než architektury R-CNN, kde byly vytvářeny navrhované oblasti, a ty pak byly postupně vyhodnocovány. Síť YOLOv1 funguje na tom principu, že obraz, který je vložen na vstup, je rozdělen na několik segmentů, které jsou porovnávány najednou. Od toho název You Only Look Once (podíváš se jen jednou) [18].

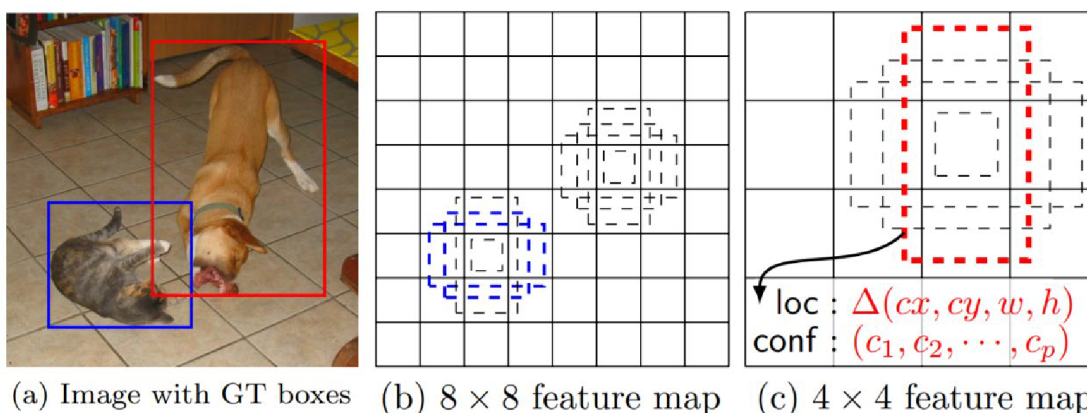
Jak ale síť přesně funguje? Nejprve je třeba popsat data, která obsahuje každá buňka. Tento vektor bude dále nazýván „V“ jako zkratka pro „vektor“. „V“ obsahuje v tomto pořadí: středovou souřadnici X objektu, středovou souřadnici Y objektu, šířku objektu, výšku objektu a pravděpodobnost objektu („confidence“). Poté ještě následuje tolik čísel, kolik typů objektů je síť schopna rozpoznat. Například pro 7 klasifikačních tříd má „V“ délku 12 hodnot. Na obrázku 1.6 je názorná ukázka toho, jak je obraz rozdělen a vyhodnocován. Nejprve je obraz rozdělen do 49 buněk, kde každá buňka obsahuje vektor „V“. Buňka, ve které se nachází střed vyhodnocovaného objektu by měla mít hodnotu pravděpodobnosti objektu 1, hodnota souřadnic X a Y by měla být velikost posunutí středu objektu vůči buňce (v rozmezí 0 až 1) a šířka a výška objektu odpovídá rozměrům objektu, kde objekty s šířkou, nebo výškou rovnající se šířce, nebo výšce

buňky mají hodnotu 1. Menší objekty mají hodnotu menší než 1 a objekty, které mají větší šířku, nebo výšku budou mít hodnotu větší než 1 [18].



Obrázek 1.6 Obraz v architektuře YOLOv1 rozdělen do mřížky 7x7 [19]

Architektura YOLOv1 však byla také několikrát modifikována. Architektura YOLOv2 (YOLO9000) v principu zůstala stejná, ale velkou novinkou této architektury jsou anchory [20]. Anchory jsou v podstatě takové šablony. Pro každou buňku jsou 3 anchory, kde jeden je ve tvaru čtverce, druhý je vysoký obdélník a třetí anchor je široký obdélník. Díky těmto anchorům se architektura lépe učí, protože jen upravuje rozměry anchoru, místo toho, aby síť každý bounding box vytvářela od nuly [15].



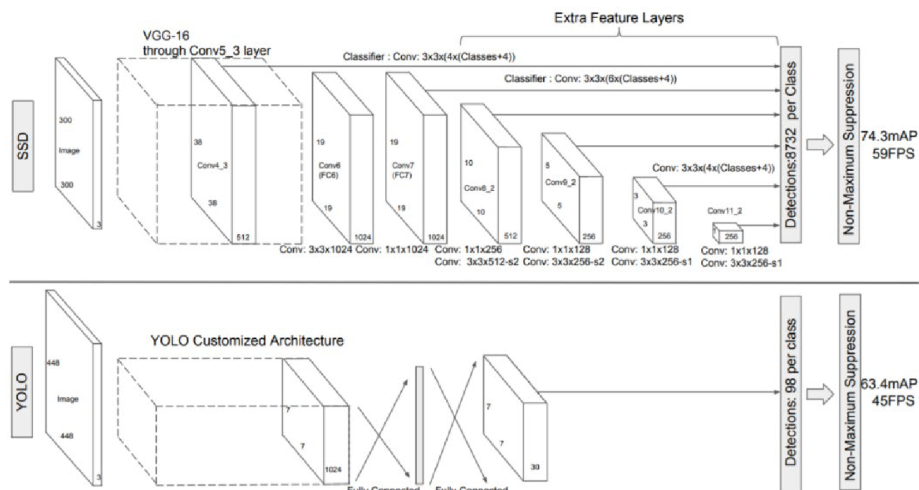
Obrázek 1.7 Ukázka funkce anchorů [21]

Poté v roce 2018 vznikla architektura YOLOv3, která přišla s další novinkou, kde obraz není rozdělen na 49 buněk (mřížka 7x7), ale na buňky o velikosti 32x32, 16x16 a 8x8 [22]. Architektury YOLO mohou v každé buňce detekovat pouze jeden objekt, a to znamená, že YOLOv1 mohla detekovat maximálně 49 objektů a zvýšení počtu buněk tedy automaticky zvyšuje maximální možný počet detekovatelných objektů v obraze [23]. Pokud bychom měli obraz o velikosti 416x416 px, tak bychom získali mřížky 13x13, 26x26 a 52x52. To znamená 3549 buněk, kde každá buňka obsahuje 3 anchory, a tím pádem 10 647 všech anchorů, kde každý anchor obsahuje bounding box (X, Y,

šířka, výška, pravděpodobnost objektu + třída objektu). Na obr. 1.7 je ukázka funkce anchorů z architektury SSD, kde anchory mají stejný význam. Architektura SSD bude popsána v další části.

1.3 SSD a RetinaNet

Ke konci roku 2016 přišla architektura s názvem SSD (Single Shot multibox Detector) [21]. Tato architektura se v mnohém podobá architekturám YOLO, a proto zde bude popsána jen krátce. SSD síť, stejně jako v YOLOv2 a YOLOv3 na začátku rozdělí obraz do různě velkých mřížek s různým počtem buněk, viz. obr. 1.7. Každá buňka obsahuje 3 anchory, kde jeden anchor je ve tvaru čtverce a dva ve tvaru obdélníku (jeden je vysoký a jeden široký) [21]. Tyto výstupy jsou následně při učení porovnávány se skutečnými výstupy. Další rozdíly, týkající se především stavby samotných sítí mezi architekturou SSD a YOLOv1 jsou na obr. 1.8. Tyto rozdíly však nejsou důležité pro pochopení principu fungování, jako spíše pro samotnou implementaci.



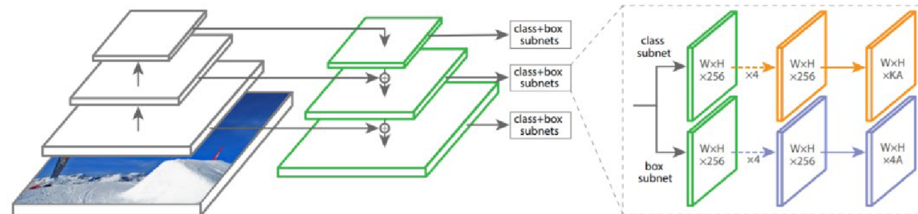
Obrázek 1.8 Ačkoliv se principem fungování může zdát, že architektury SSD a YOLO jsou stejné, není tomu tak [21]

Architektura RetinaNet je architektura, která vznikla v roce 2018 a na obr. 1.9 je zobrazen princip fungování [24]. V levé části obrázku je vidět, že se vytváří feature mapy na různých měřítkách. V druhé části se pak tyto vrstvy spojují tak, aby se poté z nich daly určit bounding boxy [25]. Tato architektura také využívá anchorů a tzv. „Focal loss“. Focal loss je vylepšená verze Cross-Entropy loss a má za úkol přidat větší váhu složitějším a důležitějším datům [26]. Chyby Cross-Entropy (CE) a focal loss (FL) jsou definovány následovnými rovnicemi, kde γ je konstanta, která se obvykle pohybuje v rozmezí 0–5 a „p“ určuje predikovanou hodnotu, pro kterou je chyba počítána [26]:

$$CE(p) = -\ln(p)\ , \tag{1.1}$$

$$\mathrm{FL(p)} = -(1-p)^{\gamma} * \ln(p)\ , \tag{1.2}$$

RetinaNet má také několik nástupců. Jsou to Retina U-Net a Light-Weight RetinaNet. Tyto architektury nijak zvlášť nemění princip fungování, ale spíše jen lehce mění některé parametry tak, aby byla RetinaNet rychlejší a přesnější [27][28].



Obrázek 1.9 Architektura RetinaNet [24]

2. VÝBĚR VHODNÉHO MODELU

Při výběru vhodného modelu budou porovnávány 2 nejdůležitější parametry. První parametr bude rychlost, kterou je schopna daná architektura zpracovat obraz. Další důležitý parametr je přesnost, se kterou dokáže daná architektura objekt rozpoznat.

2.1 Rychlost

Rychlost sítě je velmi důležitý parametr zvláště, pokud síť bude zpracovávat video, včetně „real-time“ detekce. Jelikož v zadání této práce je vytvořit model, který bude použit v průmyslu pro detekci objektů pohybujících se na dopravním pásu, bude rychlost detekce důležitá. Čím rychlejší bude zpracování jednoho snímku, tím rychleji se může potencionální dopravní pás pohybovat. Zároveň musí být splněna podmínka alespoň 20 snímků za sekundu (ideálně 30), aby byla síť vhodná pro zpracování videozáznamů. Dnešní videozáznamy totiž mají minimálně 30 snímků za sekundu, a proto bylo vhodné vybrat model, který zvládne touto rychlostí data zpracovávat.

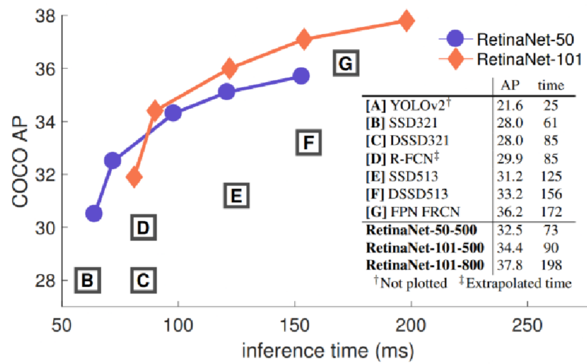
Na obr. 1.3 si lze všimnout rychlostí architektur R-CNN. Nejrychlejší z nich, architektura Faster R-CNN, dosahuje rychlosti zpracování obrazu 0,2 s, pokud nebude uvažován návrh oblasti. Čas 0,2 s znamená rychlost asi 5 snímků za sekundu. Tato rychlost je velmi malá. Architektura Mask R-CNN dosahuje stejné rychlosti, 5 snímků za sekundu [16].

Tabulka 2.1 Výsledky publikované v článku YOLOv2 (YOLO9000) na datasetu PACAL VOC2007 [20]

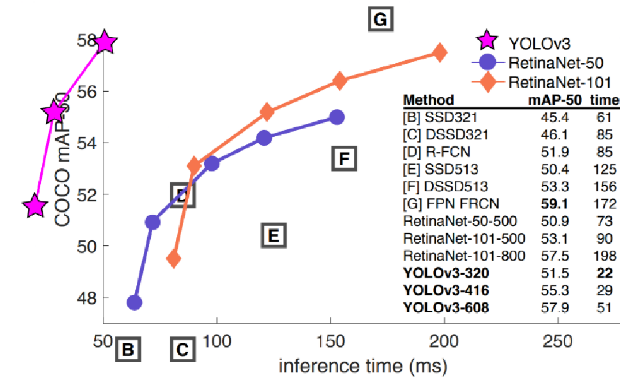
Detection Frameworks	mAP	FPS
Fast R-CNN	70.0	0.5
Faster R-CNN VGG-16	73.2	7
Faster R-CNN ResNet	76.4	5
YOLO	63.4	45
SSD300	74.3	46
SSD500	76.8	19
YOLOv2 288 x 288	69.0	91
YOLOv2 352 x 352	73.7	81
YOLOv2 416 x 416	76.8	67
YOLOv2 480 x 480	77.8	59
YOLOv2 544 x 544	78.6	40

Architektury YOLO jsou velmi rychlé. Architektura YOLOv1 dosahuje rychlosti 45 snímků za sekundu a architektura YOLOv2 rychlosti 40–91 snímků za sekundu, v závislosti na velikosti posuzovaného obrazu (od 544x544 do 288x288) [20]. Některé architektury s vysokými rychlostmi by tedy dokázaly analyzovat i video

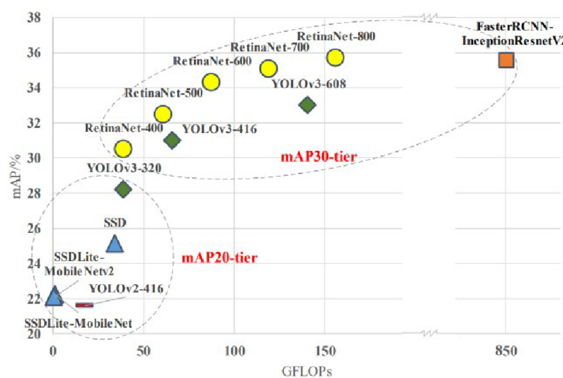
s rychlostí 60 snímků za sekundu. Architektura YOLOv3 dosahuje rychlosti 45, 34 a 19 snímků za sekundu pro obrazy o velikosti 320x320, 416x416 a 608x608 [22].



Obrázek 2.1 Výsledky publikované v článku RetinaNet na datasetu COCO [24]



Obrázek 2.2 Výsledky publikované v článku YOLOv3 na datasetu COCO [22]



Obrázek 2.3 Výsledky publikované v článku Light-Weight RetinaNet na datasetu COCO [28]

Architektura SSD dosahuje rychlostí 19-59 snímků za sekundu, opět dle velikosti obrazů a dle použitého batch size a architektura RetinaNet dosahuje spíše rychlostí

podobných architekturám R-CNN. Jedinou výjimkou je Light-Weight RetinaNet, která dosahuje podobných rychlostí, jako architektury YOLO.

V tabulce 2.1 a na obrázcích 2.1, 2.2 a 2.3 jsou zobrazeny výsledky, které byly publikovány společně s některými výše uvedenými architekturami.

2.2 Přesnost

Další velmi důležitý parametr je přesnost. Ta se měří pomocí tzv. „mean Average Precision“, případně pouze „Average Precision“ (dále mAP a AP). Tato metrika se počítá následovně. Predikované bounding boxy se přiloží na skutečné bounding boxy a všechny bounding boxy, které se překrývají s klasifikací správné třídy se označí jako „true positive“. Predikované bounding boxy, které se nepřekrývají se skutečnými bounding boxy, případně jen malým procentem plochy, se označí jako „false positive“. A nakonec všechny skutečné bounding boxy, které zůstaly nepřekryty žádným predikovaným bounding boxem se označí jako „false negative“. Poté se spočítají dvě hodnoty „Recall“ a „Precision“ dle následujících vztahů [29]:

$$Recall = \frac{True\ Positives}{Predicted\ Results} = \frac{True\ positives}{True\ positives + False\ negatives}, \tag{2.1}$$

$$Precision = \frac{True\ Positives}{Actual\ Results} = \frac{True\ positives}{True\ positives + False\ positives}, \tag{2.2}$$

Recall určuje, jaké procento skutečných bounding boxů bylo nalezeno. Precision říká, kolik z predikovaných bounding boxů bylo správně určeno. Hodnoty Recall a Precision se následně vynesou do precision-recall grafu a pokud se následně spočítá oblast pod vynesenou křivkou, vznikne hodnota AP. Pokud se tato hodnota AP spočítá pro všechny třídy a pro různá procenta, která určují true positive, nebo false positive a spočítá se jejich průměr, vznikne výsledek v podobě mAP.

V tabulce 2.1 jsou vidět výsledky mAP architektur R-CNN, YOLOv1, YOLOv2 a SSD. Nejhorších výsledků zde dosahuje YOLOv1s mAP 63,4, Fast R-CNN s mAP 70 a YOLOv2 (ve verzi 288x288) s mAP 69. Kromě těchto architektur se všechny pohybují na velmi podobných číslech. Na srovnání v obrázku 2.1 je vidět, že nejhorší výsledky zde mají architektury YOLOv2 a SSD architektury, ty ostatní mají opět podobně dobré výsledky. Na obrázcích 2.2 a 2.3 si lze také všimnout, že architektury na principu SSD architektury a architektura YOLOv2 dosahují horších přesností.

Přestože každý článek popisuje přesné výsledky svých architektur, tak nelze tyto výsledky porovnávat přímo. Pokud se však porovnají tyto čtyři grafy nepřímo, pouze relativně vůči sobě, pak lze dostat alespoň hrubý odhad. Jelikož v tomto porovnání nejsou důležitá konkrétní čísla, tak to nevadí. Relativním porovnáním lze říci, že mezi přesnější architektury se řadí architektury YOLOv3, RetinaNet, některé architektury, které vznikly modifikací architektury R-CNN nebo některé verze architektur SSD.

2.3 Rozhodnutí

V kapitolách 2.1 a 2.2 byly rozebrány dva nejdůležitější parametry, které jsou rozhodujícími při výběru sítě pro detekci objektu v obraze. Dalším důležitým parametrem při výběru by měl být i dataset, který bude využíván pro učení. Jelikož však tyto architektury byly vybrány proto, že jsou nejvíce používané, pak lze říci, že tyto architektury lze využít pro nejrůznější typy datasetů (např. COCO, Pascal VOC atd.).

V kapitole 2.1 zabývající se rychlostí těchto architektur bylo naznačeno, že v tomto směru nejvíce dominují architektury YOLO a Light-Weight RetinaNet. Pokud porovnáme tyto výsledky se závěrem v kapitole 2.2, pak lze říci, že nejvhodnější kandidáti jsou YOLOv3 a Light-Weight RetinaNet. Tyto architektury dosahují vysokých rychlostí a jejich úspěšnosti jsou také relativně dobré.

V této práci bude použita architektura YOLOv3, protože tato architektura je používanější [9]. To v sobě zahrnuje několik dalších potencionálních výhod oproti architektuře Light-Weight RetinaNet. První potencionální výhoda je, že tato architektura je více prověřená společností, a tudíž spolehlivější při dosahování výsledků. Druhá potencionální výhoda je ta, že tato architektura je o něco starší, což může napovídat tomu, že je architektura jednodušší jak pro pochopení, tak i pro samotnou implementaci.

3. DATASET

Další důležitá součást projektu je dataset, na kterém se bude síť učit. Při výběru datasetu existují tři možnosti postupu. První možnost je použít již existující dataset, druhá je vytvoření vlastního datasetu a třetí možnost je již existující dataset rozšířit o vlastní data. Výhoda použití již existujícího datasetu je v tom, že ušetří čas a není nutné vytvářet prostředí, ve kterém fotografie vznikají. Výhodou vytvoření vlastního datasetu je zase v tom, že si lze přesně určit podmínky, za kterých budou fotografie pořizovány, například pozadí, jednotnost a kvalita. Pro ty, kteří žádný dataset nikdy nevytvářeli je další výhoda v tom, že si prakticky vyzkouší, jak se takovýto dataset vytváří. V této práci tedy dataset byl vytvářen od začátku.

Ačkoliv byl dataset v této práci vytvářen pro konkrétní architekturu YOLOv3, která má čtvercový vstup, tak fotografie v tomto datasetu nebyly čtvercové z toho důvodu, že většina fotografií nemá čtvercový poměr stran. Fotoaparát, který byl využit pro fotografování fotografií do datasetu také vytváří pouze obdélníkové fotografie. Protože model musí být vhodný k obecnému použití, bylo nutné stejně implementovat funkci, která převede obdélníkové fotografie různých velikostí na čtvercové fotografie o stejné velikosti.

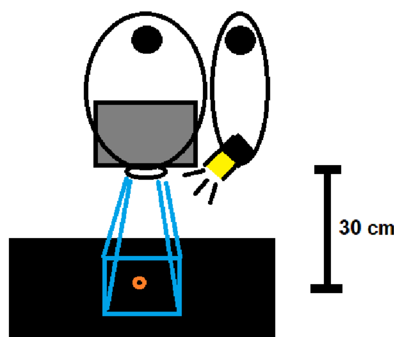
3.1 Přípravy

3.1.1 Nastavení scény

Protože byl dataset složen z fotografií, bylo nutné vytvořit scénu a obstarat si kvalitní fotoaparát na kterém lze nastavit velké množství parametrů. Jelikož zadání říká, že se musí jednat o detekci objektů z průmyslu, musela být scéna co nejvíce podobná průmyslovému prostředí. Průmyslovým prostředím zde byl myšlen dopravní pás, na kterém se objevují různé objekty. Dopravní pás bývá obvykle černý, a proto bylo ve scéně použito černé pozadí. Na obrázku 3.1 lze vidět, jak byla scéna vytvořena. Fotoaparát byl zavěšen pomocí provázků asi 30 cm nad podložkou tak, aby objektiv mohl směřovat kolmo dolů. Vedle fotoaparátu byla ještě zavěšena LED žárovka s zářivostí 950 lm a bílou barvou, opět ve výšce asi 30 cm nad podložkou.

Podložka, na kterou byly pokládány předměty byla černá textilie. Jako fotoaparát byl použit fotoaparát Canon EOS 1000D s nastavenou clonou f/10, ISO bylo nastavené na hodnotu 200 a čas expozice byl 0,05 sekundy. Tyto hodnoty byly nalezeny metodou „pokus omyl". Na počátku byla nastavena clona f/2, ale předměty v obraze vypadaly kvůli nízké hloubce ostrosti rozmazaně, a proto bylo nutné tuto hodnotu zvýšit. Tato hodnota však nemohla být příliš vysoká, protože pak by byl obraz příliš tmavý a bylo by nutné prodloužit čas expozice, nebo použít silnější osvětlení. Jednodušší bylo zvýšit čas expozice, jenže tato doba nesměla být ani moc dlouhá, aby bylo možné vyfotit co nejvíce fotografií za co nejkratší čas. Nakonec clona f/10 a expozice 0,05 sekundy se jevila jako

nejlepší kombinace. Hodnota ISO byla už od začátku nastavena na 200, protože pokud by byla tato hodnota vyšší, ve fotografiích by byl zbytečný šum.



Obrázek 3.1 Náčrt fotografovací scény

3.1.2 Klasifikační třídy

Během fotografování bylo zjištěno, že plocha, kterou fotoaparát snímá byla velká asi 400 cm^2. Předměty, které budou zahrnuty v datasetu tím pádem nesměly být v nejširším místě širší, než asi 20 cm. Malé předměty nejrůznějších tvarů jsou kancelářské potřeby. Proto předměty zahrnuté v tomto datasetu jsou **gumičky**, **kolíčky**, **připínáčky**, **párátka**, **sirky**, kancelářské **sponky** a připínáčky ve tvaru vlajky, které dále budou nazývány „**vlaječky**".

V nejpoužívanějších datasetech, jako je COCO nebo Pascal VOC se každá třída vyskytuje v řádech tisíců, případně desetitisíců [30]. V tomto datasetu byl každý předmět vyfocen tisíckrát. Toto číslo bylo vybráno náhodně s vědomím, že všechny tyto fotografie budou augmentovány dvacetkrát. Samozřejmě bylo nutné každý předmět vyfotit více než tisíckrát, protože se mezi fotkami mohly vyskytnout fotografie, které byly špatně vyfocené.

3.2 Úprava fotografií

Poté, co bylo vyfoceno více než tisíc fotografií od každého předmětu, bylo nutné smazat fotografie, kde předmět zasahoval mimo záběr, kde byla omylem vyfocena ruka, anebo ty, které byly rozmazané. Ze zbylých fotografií bylo náhodně vybráno 1000, které byly dále používány.

K dalším úpravám již byla využita výpočetní technika. V programovacím jazyku Python byly vytvořeny krátké skripty, které jsou přiloženy v příloze A, v souboru „IMG_PREPROCESS.py". Tyto skripty byly nejprve využity k zamíchání fotografií, přejmenování těchto fotografií na daný tvar „třída_číslo.jpg" (např. „gumicka_420.jpg"), a také ke zmenšení těchto fotografií z původní velikosti 1936x1288 px na novou velikost 600x400 px kvůli redukci objemu dat (funkce „rename_resize"). Původní velikost jedné

fotografie se pohybovala kolem 800-950 kB a díky tomuto zmenšení se velikost jedné fotografie pohybovala kolem 60-65 kB. Lze tedy hovořit o čtrnáctinásobném snížení objemu dat. Předměty byly na fotografiích stále dobře rozlišitelné (viz. obr. 3.2).

Obrázek 3.2 Ukázka fotografie připínáčku s rozlišením 600x400 px

Druhou možností, jak zmenšit fotografie bylo tyto fotografie oříznout. Pokud by však tyto fotografie byly oříznuty, pak by se znovu objevil problém, kde by předměty zasahovaly mimo obraz, nebo by dokonce byly z fotografie vyříznuty úplně. Ačkoliv se zmenšením fotografie ztratily některé informace, pokud je v tomto stylu modifikován celý dataset, pak není žádná fotografie zvýhodněna nebo znevýhodněna.

3.3 Anotace a augmentace

Další krok bylo vytvoření anotace k těmto fotografiím. K anotaci byl využit program VGG Image Annotator [31]. V tomto programu šlo anotování rychle a jednoduše. Výsledné anotace byly ve tvaru [X, Y, šířka, výška], kde X a Y = 0 odpovídá levému hornímu rohu a X = 599 a Y = 399 pravému dolnímu rohu. Šířka a výška je udávána v pixelech. Důležité je zmínit, že architektura YOLOv3 pracuje se souřadnicovým systémem, který hodnotou X a Y myslí střed objektu, nikoliv levý horní roh, a proto musely být tyto hodnoty ještě přepočítány následovně:

```
new_x = x + (width / 2)
new_y = y + (height / 2)
```

Po vytvořené anotaci přišla na řadu augmentace. K augmentaci byl opět využit programovací jazyk Python, díky čemuž byl celý proces augmentování zjednodušený a zautomatizovaný (funkce „augment“). V této fázi dataset obsahoval 7 tříd, kde každá třída obsahovala 1000 fotografií. Nejprve byly fotografie augmentovány rotací o 180°. Pokud by dataset obsahoval fotografie s poměrem 1:1, pak by se daly využít i rotace o ± 90°. U obdélníkových fotografií by došlo k nežádoucí deformaci. Poté byly fotografie ještě převráceny horizontálně a vertikálně. V této fázi každá třída obsahovala 4000 fotografií.

Na těchto 4000 fotografií byly následně aplikovány další 4 druhy augmentace. První augmentace byla augmentace rozmazáním. K tomuto účelu byla využita funkce „blur" z knihovny OpenCV, kde argumentem bylo náhodně velké jádro „kernel" od 2x2 do 4x4 (včetně obdelnikových kernelů např. 4x2). Druhá augmentace byla augmentace změnou jasu. Opět byla využita knihovna OpenCV, kde fotografie byla nejprve převedena z RGB (červená, zelená, modrá) kódování do HSV (odstín, saturace, jas). V tomto kódování byla měněna hodnota jasu o náhodnou hodnotu od -35 do +35. Poté byly fotografie opět převedeny do RGB kódování. Třetí augmentace byla augmentace, která posunula barevné spektrum fotografie. Náhodně vybrané barevné kanály byly vždy posunuty o náhodnou hodnotu od 10 do 35. Poslední augmentace byla augmentace typu „salt & pepper", kde byly náhodné pixely „přebarveny" na bílo, nebo na černo. Implementace, která je přiložena v příloze, změní barvu náhodně 0-10 % pixelů. Díky všem těmto augmentacím měla každá třída v této fázi 20000 fotografií.

Jakmile byly augmentovány fotografie, bylo nutné ještě augmentovat i anotace k těmto fotografiím (funkce „anotate"). Anotace byly přepočítány u všech fotografií, které byly otočeny, nebo převráceny. Například anotace pro fotografie otočené o 180° byly souřadnice přepočítány takto:

```
new_x = img_width – old_x – width
new_y = img_height – old_y – height
```

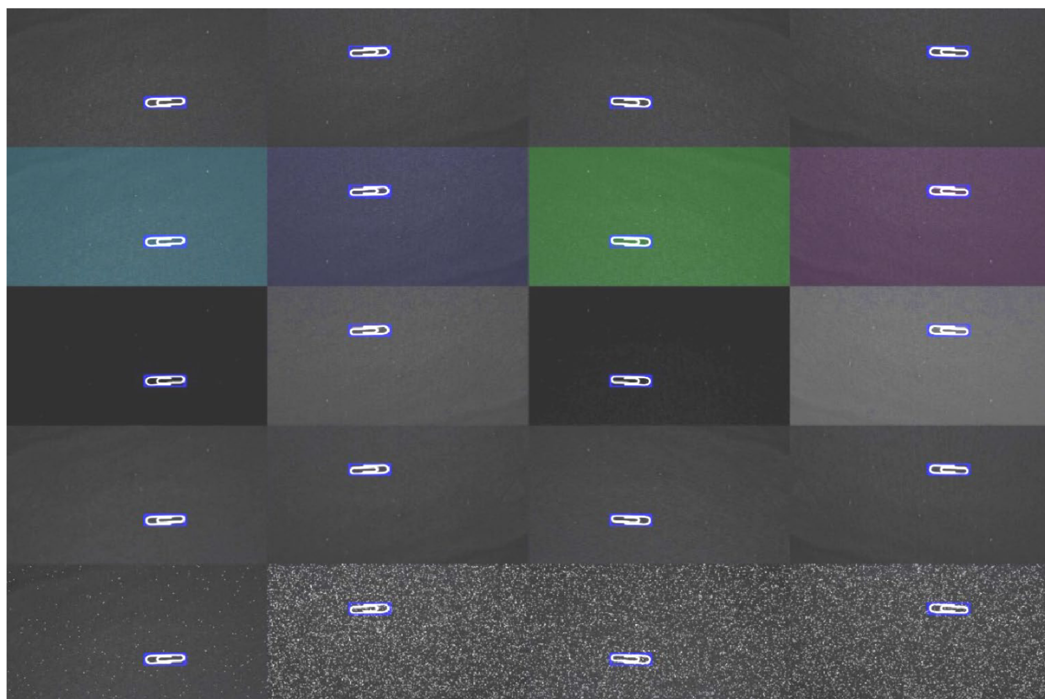
Protože by se mohlo stát to, že v trénovacím datasetu bude např. obrázek „gumicka_vFlip_66.jpg" a ve validačním nebo testovacím datasetu by se vyskytl např. obrázek „gumicka_66.jpg", který je velmi podobný, mohlo by se stát, že validační, potažmo testovací výsledky by byly zkreslené. Bylo proto nutné ještě doplnit funkci „sort_dataset", která trénovací, validační a testovací datasety protřídila tak, aby všechny augmentace jednoho obrázku se vyskytovaly pouze v trénovacím, validačním, respektive testovacím datasetu.

3.4 Popis výsledného datasetu

Výsledný dataset tedy obsahoval 20 000 fotografií o rozměrech 600x400 px, kde 1000 fotografií byly originální, 5000 bylo otočeno o 180°, 5000 převráceno vertikálně a 5000 převráceno horizontálně. Dataset obsahoval 7 tříd, a tím pádem celý dataset obsahoval 140 000 fotografií. Celková velikost všech fotografií byla 10,2 GB. Všechny fotografie byly uloženy pod názvem ve tvaru „třída_augmentace1_augmentace2_číslo.jpg" (např. paratko_brightness_vFlip_489.jpg, nebo pripinacek_hFlip_929.jpg). Celý dataset byl rozdělen tak, že 70 % obrázků (98 000) bylo označeno jako trénovací dataset a zbylých 42 000 obrázků bylo rozděleno mezi validační a trénovací dataset napůl, tedy 21 000 obrázků každý.

Ukázka všech augmentací je na obr. 3.3. Vlevo nahoře je originální fotografie a vpravo od ní je rotace o 180°, horizontální převrácení a vertikální převrácení. Pod těmito fotografiemi jsou augmentace těchto fotografií barevným posunutím, změnou jasu,

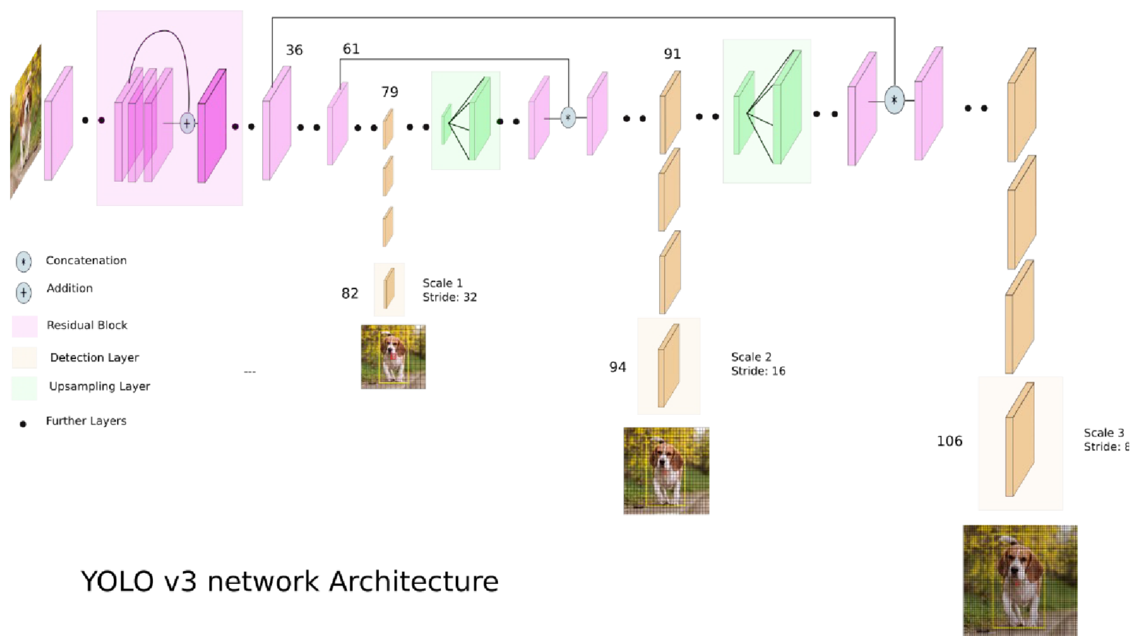
rozmazání (blur) a v poslední řadě je augmentace salt & pepper. Na obrázcích si lze všimnout i modrých rámečků, které odpovídají anotacím.



Obrázek 3.3 Příklad anotace a augmentace na fotografii kancelářské sponky

4. IMPLEMENTACE

Celý model byl implementován v jazyce Python, celý kód je přiložen v příloze A. Hlavní knihovnou pro implementaci se stala knihovna PyTorch, což je jedna z nejpopulárnějších knihoven pro účely strojového učení [32]. Tato knihovna byla upřednostněna před ostatními knihovnami díky tomu, že nabízí spoustu funkcí, a zároveň je intuitivní a jednoduchá na použití.



Obrázek 4.1 Demonstrační model architektury YOLOv3, kde jsou důležité 3 výstupy. Výstup 1 (scale 1) je rozdělen do mřížky s buňkami o velikosti 32x32 px, výstup 2 (scale 2) obsahuje buňky o velikosti 16x16 px a poslední výstup (scale 3) obsahuje buňky o velikosti 8x8 px [33]

Na obr. 4.1 je zobrazena struktura architektury YOLOv3. Úplně vlevo je vidět obrázek vkládaný na vstup. Ten nejprve prochází Darknetem-53 (fialové vrstvy ve světlejším fialovém obdélníku), který byl v článku YOLOv3 vybrán, jako nejvhodnější [22]. Darknet-53 je tzv. „feature extractor“ (někdy nazývaný také „backbone“). Jako feature extractor se obvykle používají architektury pro klasifikaci obrazu (např. ResNet, VGG-16 [34]) a mají za úkol určitým způsobem předzpracovat obraz tak, aby v něm „vynikly“ důležité vlastnosti fotografie. Po Darknetu-53 pak následují další různé vrstvy, které budou blíže popsány v další kapitole.

4.1 YOLOv3

Jako první byla naimplementována samotná architektura YOLOv3. Na obr. 4.1 je vidět pouze orientační podoba celé architektury, ale nenese v sobě žádné konkrétní hodnoty.

Na obr 4.2 již je přesně nadefinována alespoň první část této architektury – Darknet-53. Je zde vidět, že bylo nutné naimplementovat několik konvolučních vrstev, ale i residuální skoky. Jelikož je však Darknet-53 pouze feature extractor, poslední tři vrstvy byly odstraněny a místo nich byla napojena další část YOLOv3. Nejen Darknet-53, ale i zbytek architektury je nadefinován na githubovém účtu autora YOLOv3, Josepha Redmona („pjreddie“) [35]. Dle této definice byla celá síť naimplementována. První rozdělení se nachází pár konvolučních vrstev za Darknetem-53. Za tímto rozdělením se nachází dalších pár konvolučních vrstev a na konci se nachází první výstup. Po prvním rozdělení v hlavní větvi se nachází upsampling („zvýšení rozlišení“), za kterým následuje spojení s předešlými hodnotami zakončené druhým rozdělením. Na konci druhé větve se nachází druhý výstup. Třetí výstup pak je na konci druhé větvě druhého rozdvojení, která opět obsahuje upsample a spojení s předešlými hodnotami (viz. obr. 4.1).

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Obrázek 4.2 Darknet-53 [22]

V originálním článku byly představeny YOLOv3 architektury se třemi různě velkými vstupy. Na vstupu tedy mohou být fotografie o velikosti 608x608, 416x416, nebo 320x320 px [22]. Větší rozlišení je vhodné pro větší přesnost, a naopak menší rozlišení je dobré pro rychlost sítě. V této implementaci byla zvolena velikost vstupu 320x320 px právě kvůli své rychlosti. Na výstupu sítě jsou 3 mřížky, kde každá jedna buňka této mřížky má velikost 32x32, 16x16 a 8x8 (viz. obr. 4.1). Každá buňka v sobě navíc musí obsahovat 3 anchory a každý anchor v sobě musí obsahovat informace o případném bounding boxu. Na výstupu má tedy první výstup rozměr matice 10x10x3x12, druhý výstup 20x20x3x12 a třetí výstup 40x40x3x12. Poslední rozměr matice má velikost 12, protože 4 hodnoty zde zastupují bounding box (pozice x, pozice y, šířka, výška), páté číslo značí pravděpodobnost objektu v daném anchoru (také tzv. „confidence“) a dalších sedm hodnot ukazuje, který předmět se v anchoru vyskytuje (pokud se nějaký vyskytuje).

Ve výsledku tedy YOLOv3 vrací 6300 anchorů, kde každý obsahuje 12 hodnot, které bylo nutné dále zpracovat.

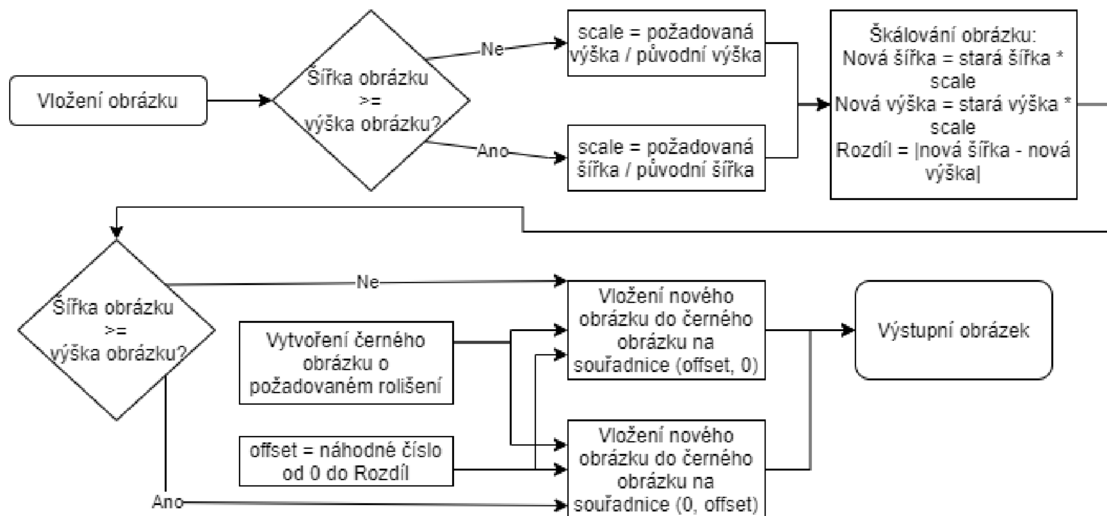
4.2 Dataset

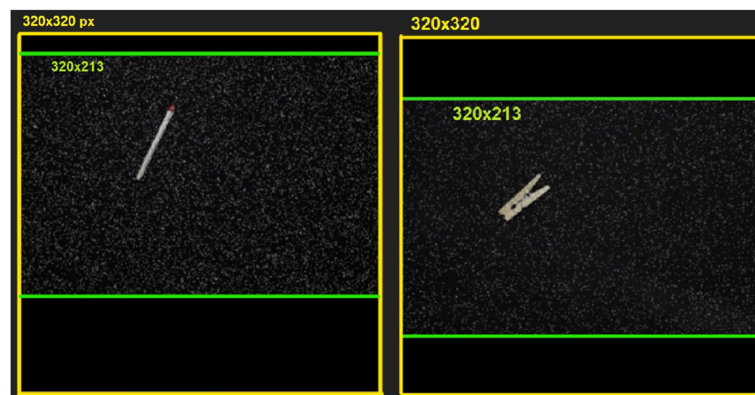
Další významný bod je načítání dat z datasetu. Pro tento účel byla využita PyTorch třída DataLoader.

```
train_dataset = ImageLoader(config[11] + "/train.csv", classes,
config)
train_loader = DataLoader(train_dataset, config[2], True,
pin_memory=config[9], num_workers=config[10])
```

Tato funkce zajišťuje načítání vzorků a obsahuje různé přepínače pro jednodušší manipulaci s daty. Například v této implementaci byly využity přepínače batch size, shuffle, „pin memory“ [36] a „num workers“ [37]. Jeden z argumentů třídy DataLoader musí být třída, která DataLoaderu poskytuje data. V této implementaci byla tato třída pojmenována ImageLoader. Příklad použití třídy ImageLoader:

```
dataset = ImageLoader(**kwargs)
image, bounding_box = dataset[0]  # dataset[1], dataset[2]…
```





Obrázek 4.3 Vývojový diagram úpravy obrázků a 2 ukázky výstupních obrázků

Tato třída obsahuje metodu “__getitem__”, která nejprve načte příslušnou fotografii dle požadovaného indexu, a pokud tato fotografie nemá požadovanou velikost (v tomto případě 320x320 px), musí být upravena. Úprava probíhá tak, že se nejprve fotografie zmenší (případně zvětší) tak, aby největší rozměr fotografie byl 320 px. Poté se vygeneruje černý čtvercový obrázek s rozměry 320x320 px a pak je fotografie vložena na náhodné místo. Na obr. 4.3 je tento proces detailněji zobrazen, včetně výsledných obrázků. Žlutý čtverec ohraničuje černý obrázek s rozměry 320x320 px a zelený čtverec ohraničuje fotografii, která byla zmenšena z 600x400 px na rozměr 320x213 px. Každá fotografie je následně vložena na různá místa (v tomto případě do různé výšky).

Po této transformaci fotografií bylo nutné ještě upravit bounding boxy tak, aby odpovídaly objektu v obraze i po transformaci. Při načítání bounding boxů jsou také souřadnice a rozměry převedeny na interval <0; 1>, což odpovídá relativním hodnotám. Následující rovnice popisuje tento přepočet, kde proměnná P označuje původní souřadnici x, y, šířku a výšku a N označuje novou hodnotu x, y, šířky a výšky.

$$N = \frac{P * scale}{Nová\ šířka(výška)\ obrázku\ (šířka == výška)} \quad (4.1)$$

Protože YOLOv3 vrací mřížky anchorů a tyto výstupy musí být porovnány se skutečnými hodnotami kvůli učení, musí být tyto bounding boxy následně převedeny na mřížkový tvar, který dále bude nazýván „targety“.

4.3 Target conversion

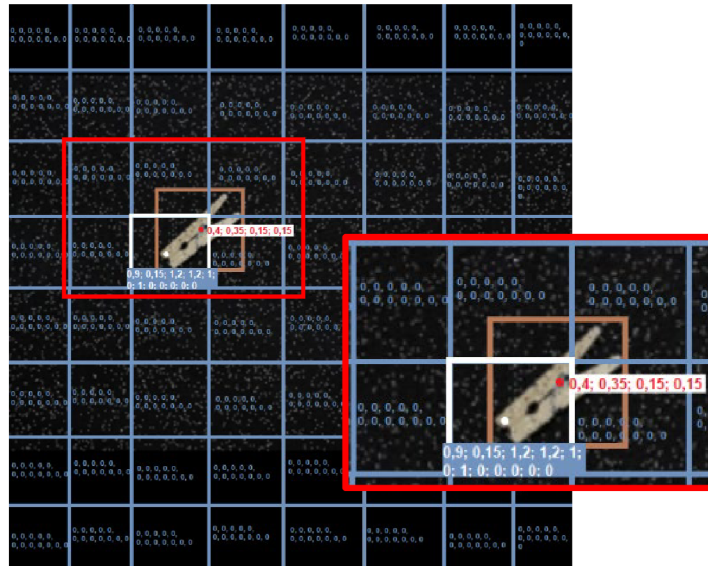
V této kapitole budou popsány funkce, které bounding boxy převádí na targety a targety pak zpátky na bounding boxy. Funkce boxes_to_targets funguje na jednoduchém principu. Nejprve se vytvoří prázdné mřížky targetů. Poté je posuzovaný bounding box porovnán pomocí IoU s anchory tak, že například čtvercový anchor by měl být přiřazen k objektům, které mají bounding boxy blízký čtverci. Následně se v každém měřítku podle pozice bounding boxu nastaví zjištěný anchor v korespondující buňce na hodnoty bounding boxu. Hodnoty x, y, šířka a výška jsou upraveny tak, aby byly relativní vůči buňce. Pokud jsou objekty širší nebo vyšší než buňka, je tato hodnota větší než 1, zatímco hodnoty x a y můžou nabývat pouze hodnot 0 až 1. Výsledek je zobrazen na obr. 4.4.

Funkce targets_to_boxes je reverzní funkce k funkci boxes_to_targets a slouží k tomu, aby bylo možné výsledky, které se objeví na výstupu YOLOv3, zobrazit ve tvaru bounding boxů. V této funkci jsou všechny anchory ve všech buňkách, které mají nenulovou pravděpodobnost objektu převedeny do tvaru bounding boxu relativnímu vůči obrázku. Na následující rovnici je tento přepočet nadefinován.

$$X = \frac{X_0 + A}{scale}, Y = \frac{Y_0 + B}{scale}, \text{Šířka} = \frac{Šířka_0}{scale}, Výška = \frac{Výška_0}{scale} \quad (4.2)$$

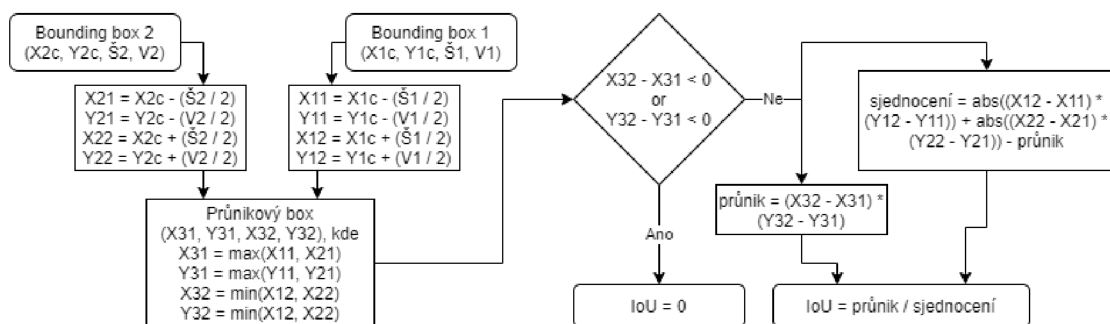
Hodnoty s indexem 0 značí původní hodnoty, A a B označují souřadnice buňky v targetové mřížce a scale zde označuje rozlišení targetové mřížky (např. pro 10x10 mřížku se scale = 10).

Výsledkem pro vstup o rozlišení 320x320 px je 6300 bounding boxů, které byly následně zredukovány pomocí funkce Non-max suppression.



Obrázek 4.4 Transformace z tvaru bounding boxu (hnědá, červená) na tvar targetů (bílá, bledě modrá. Nenulové hodnoty jsou zde jen orientační a neodpovídají skutečným spočteným hodnotám.

4.4 Intersection over Union

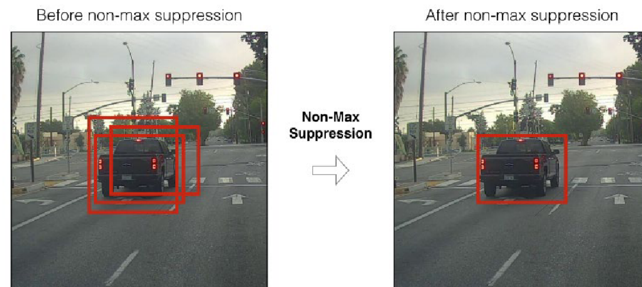


Obrázek 4.5 Vývojový diagram výpočtu IoU, kde x a y vstupních bounding boxů vyjadřuje pozici středu objektu

Tato funkce je v oblasti detekce objektu jedna z nejjednodušších, ale zároveň nejdůležitější, protože je využívána například pro NMS, výpočet chyby sítě atd. Jedná se o funkci, která porovnává dva bounding boxy a výsledkem je podíl překrytí, vůči sloučení (0 pro nepřekrývající se a 1 pro 100% překrývající se bounding boxy). Podrobný

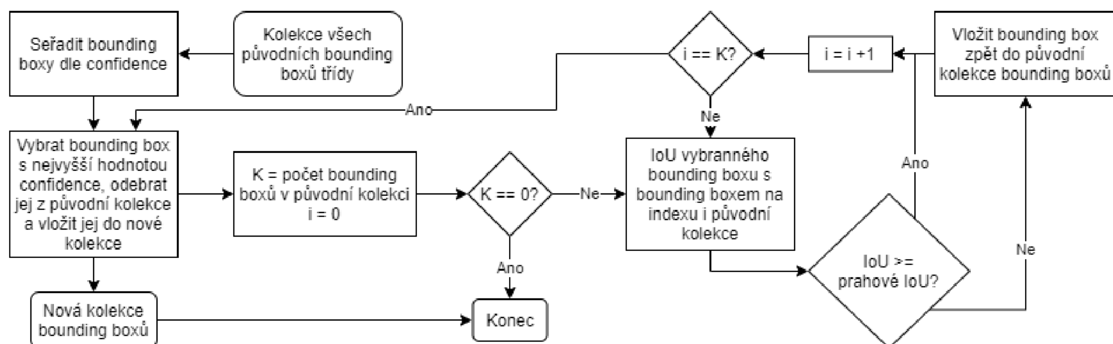
postup výpočtu je zobrazen na obr. 4.5. Nejprve musí být vytvořen průnikový box, pak se vypočítá jeho obsah. Následně se tento obsah průnikového boxu podělí obsahem obou původních bounding boxů, od kterých se ještě předtím odečte obsah průnikového boxu. Výsledkem tohoto podílu je IoU. Nejčastěji se říká, že pokud je hodnota IoU $\geq$ 0,5, jedná se o dobrou predikci [38].

4.5 Non-max suppression



Obrázek 4.6 Non-max suppression [39]

Na výstupu funkce targets_to_boxes je velké množství bounding boxů a velké procento z nich se překrývá (viz. obr. 4.6). Princip této funkce je následující. Nejprve jsou smazány všechny bounding boxy, které obsahují velmi nízkou pravděpodobnost objektu. Zbylé bounding boxy se následně seřadí sestupně podle pravděpodobnosti výskytu objektu (pátá hodnota v anchoru). S bounding boxem s nejvyšší hodnotou pravděpodobnosti se poté porovnají všechny další bounding boxy pomocí IoU. Pokud výsledné IoU je větší, než určitá hodnota (v této implementaci 0,5), pak je bounding box s nižší pravděpodobností smazán. Zbylé bounding boxy se znovu seřadí sestupně, první bounding box se porovná s ostatními a ty, které mají větší hodnotu IoU, než je zadáno, jsou smazány. Tento proces je opakován do té doby, než jsou smazány všechny „nadbytečné" bounding boxy [39] (obr. 4.7). V implementaci bylo nutné ošetřit, aby byly porovnávány pouze bounding boxy stejné třídy.



Obrázek 4.7 Vývojový diagram procesu NMS

4.6 Mean Average Precision

Funkce mean average precision je funkce, která slouží pouze k hodnocení architektury a její princip je již přiblížen v kapitole 2.2. V této implementaci byla mAP počítána tak, že se každý predikovaný bounding box porovnal s ground truth bounding boxy pomocí IoU. Pokud IoU bylo větší, než zadaná hodnota (v implementaci hodnota 0,5), tak se jednalo o true positive. Ground truth bounding boxy, ke kterým nebyl nalezen žádný predikovaný bounding box se označily jako false negative a všechny predikované bounding boxy, ke kterým nebyl přiřazen žádný ground truth bounding box se označily jako false positive [29].

Následně se všechny bounding boxy seřadily podle pravděpodobnosti předmětu v bounding boxu a podle vzorců 2.1 a 2.2 se spočítaly hodnoty recall a precision. Tyto hodnoty byly následně vyneseny do Precision-recall grafu a vypočítala se velikost oblasti pod touto křivkou. Tyto kroky se provedly pro každou třídu a případně i pro různé hodnoty IoU, které určují, jestli se jedná o TP bounding box, nebo FP a vypočítal se jejich průměr [29]. Tato výsledná hodnota se nazývá mAP.

4.7 Loss funkce

Loss funkce je funkce, která má za úkol porovnat 2 sady hodnot a vrátit číselnou hodnotu reprezentující chybu. První sada hodnot obvykle představuje skutečné hodnoty (ideální) a druhá sada hodnot jsou predikované hodnoty neuronové sítě. Čím menší je chybová hodnota, tím bližší jsou predikované hodnoty skutečnosti. Loss funkce v této implementaci byla vytvořena dle článků YOLO [20][22]. Zde se počítají celkem čtyři chyby, které jsou následně sečteny do jedné hodnoty. Funkce BCE byla použita pro klasifikační chyby (kde výstup by měl být 0, nebo 1) a funkce MSE pro hodnoty, které mohly nabývat ostatních hodnot. Na následujících rovnicích jsou tyto chyby zadefinovány:

$$box_loss = MSE\big(\big(sigmoid(x_{GT}, y_{GT}), anchor_{(W,H)} * e^{(w_{GT}, h_{GT})}\big), (x_P, y_P, w_P, h_P)\big), \tag{4.3}$$

$$conf_loss = MSE\big(IoU(box_{GT}, box_P) * c_{GT}, sigmoid(c_P)\big), \tag{4.4}$$

$$no_conf_loss = BCE(c_{GT}, c_P), \tag{4.5}$$

$$class_loss = BCE(class_{GT}, class_P), \tag{4.6}$$

kde MSE je tzv. „mean squared error“, BCE je PyTorch funkce „BCEWithLogitsLoss“, GT značí skutečnost, P predikci, x a y jsou souřadnice a w a h značí hodnoty šířky a výšky.

Box_loss je chyba, která počítá chybu umístění a velikost predikovaného bounding boxu, conf_loss odpovídá za hodnotu confidence. Predikovaná hodnota confidence by zde měla odpovídat hodnotě IoU skutečného a predikovaného bounding boxu [40]. No_conf_loss je chyba vypočtená porovnáním confidence všech buněk, které by měli mít hodnotu 0 a class_loss je druhá klasifikační chyba, která má za úkol vypočítat chybu určení třídy.

4.8 Config, train a show image

Nyní byly popsány všechny důležité funkce a je třeba jim dát strukturu. V celém programu figuruje proměnná config. Tato proměnná je list zapouzdřující všechny různé konstanty, kterými lze jednoduše upravovat chratakter celého programu. Config obsahuje learning rate, weight decay (tyto 2 hodnoty byly nadefinovány společně s architekturou YOLOv3), batch size, četnost validace, počet epoch k trénování, velikost vstupu, mAP thresholds, nms threshold, boxes to targets threshold, pin memory, num workers (tyto 2 hodnoty slouží pro načítání datasetu), složky s fotografiemi a anotacemi, rohový/středový systém souřadnic, trénování, načítání, přepínač ukazování testovacích obrázků, zamíchání dat v datasetech a jméno pro ukládání naučených vah.

Funkce train do této konfigurace vkládá ještě informaci o případné grafické kartě, anchory, velikost výstupní matice a anchory zmenšené na velikost buněk. Ve funkci train je nainicializována také proměnná „history“, která v sobě zahrnuje celou historii učení (průměrnou hodnotu loss při jednotlivých epochách, validační loss hodnoty a validační hodnoty mAP). Po této proceduře se již nainicializuje samotný model, optimizer a loss funkce:

```
model = YoloV3.Architecture(classes).to(config[20])
optimizer = torch.optim.Adam(model.parameters(), lr=config[0],
weight_decay=config[1])
loss_fn = YoloV3.LossFunction()
scaler = torch.cuda.amp.GradScaler()
```

Pokud je požadováno načtení již dříve naučených vah, následuje kód, který obstarává načítání:

```
checkpoint = torch.load(config[15])
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
history = checkpoint['history']
```

Následuje načítání datasetů, kde „function“ nabývá hodnot „train“ pro trénovací průchod, „val“ pro validační průchod a „test“ pro testovací průchod:

```
function_dataset = ImageLoader(config[11] + "/function.csv", classes,
config)
function_loader = DataLoader(function_dataset, config[2], config[17],
pin_memory=config[9], num_workers=config[10])
```

V této chvíli jsou již nainicializovány všechny důležité objekty a nyní lze přejít k učení, validaci, nebo jen k testování pomoci funkce „play“:

```
model, history = play(model, optimizer, loss_fn, scaler, config,
classes, function_loader, history, “function”)
```

Funkce play má základní posloupnost funkcí, která je shodná pro trénink, validaci i testování a podle toho, co je zrovna potřeba se přidávají další funkce. Nejprve se do modelu vloží obrázek, a ten vrátí výsledné predikce. Tyto predikce se pak porovnají se skutečnými výsledky. Hodnota chyby byla vypočtena následovně:

$$loss = loss_fn_{(GT_{S1}, P_{S1})} + loss_fn_{(GT_{S2}, P_{S2})} + loss_fn_{(GT_{S3}, P_{S3})}, \tag{4.7}$$

kde Sn odpovídá konkrétním škálovým mřížkám (targetům). Pokud se jedná o učící průchod funkcí play, je nutné upravit váhy z vypočtené chyby:

```
scaler.scale(loss).backward()
scaler.step(optimizer)
scaler.update()
```

Pokud se nejedná o učící proces, predikce se převedou z targetů do bounding boxů a provede se non-max suppression:

```
true_boxes = targets_to_boxes(y, config[21])
pred_boxes = targets_to_boxes(out, config[21])
all_true_boxes.append(nms(true_boxes, config[7]))
all_pred_boxes.append(nms(pred_boxes, config[7]))
```

A pokud se jedná o testovací obrázek, lze si obrázek vykreslit společně s bounding boxy funkcí „show_image“:

```
show_image(x.to('cpu'), all_true_boxes[-1], all_pred_boxes[-1],
classes)
```

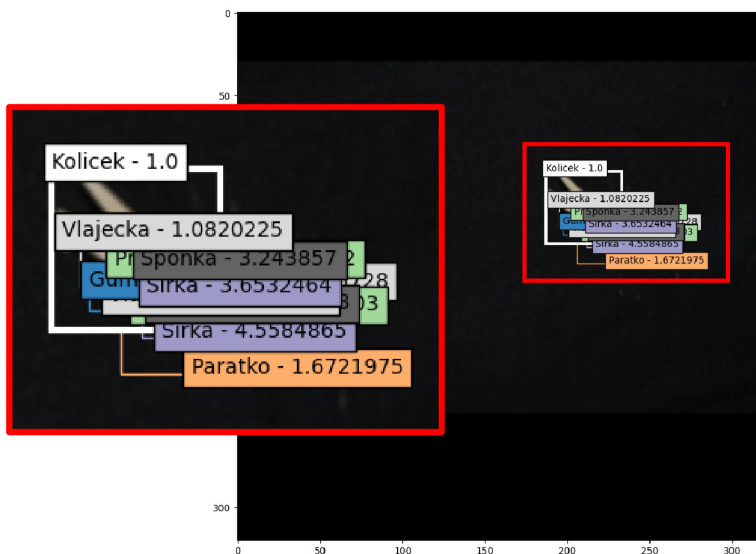
Na konci každé validace se model uloží do složky „checkpoints“:

```
path = "checkpoints/" + config[18] + "_" + str(sum(losses) /
len(losses)) + ".pth.tar"
torch.save({'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'history': history},
           path)
```

A na konci testovacího běhu se zobrazí veškerá historie učení a validace:

```
axs[0].plot(history['train_losses'], 'r-x', label="Train losses")
axs[0].plot(history['val_losses'], 'b-x', label="Validation losses")
axs[1].plot(history['val_maps'], 'g-x', label="Validation mAPs")
plt.show()
```

Funkce show_image má za úkol zobrazit obrázek společně s predikovanými bounding boxy, a pro porovnání, i se skutečnými bounding boxy (viz. obr. 4.8). Na obrázku 4.8 je zobrazena třída bounding boxu i jeho confidence (pravděpodobnost). Obrázek 4.8 je také důkazem funkční implementace. Stačilo modelu předložit jen pár trénovacích fotografií a predikce již vykazovala výsledky, které byly smysluplné. Nyní bylo nutné model naučit na úplném datasetu z kapitoly 3.

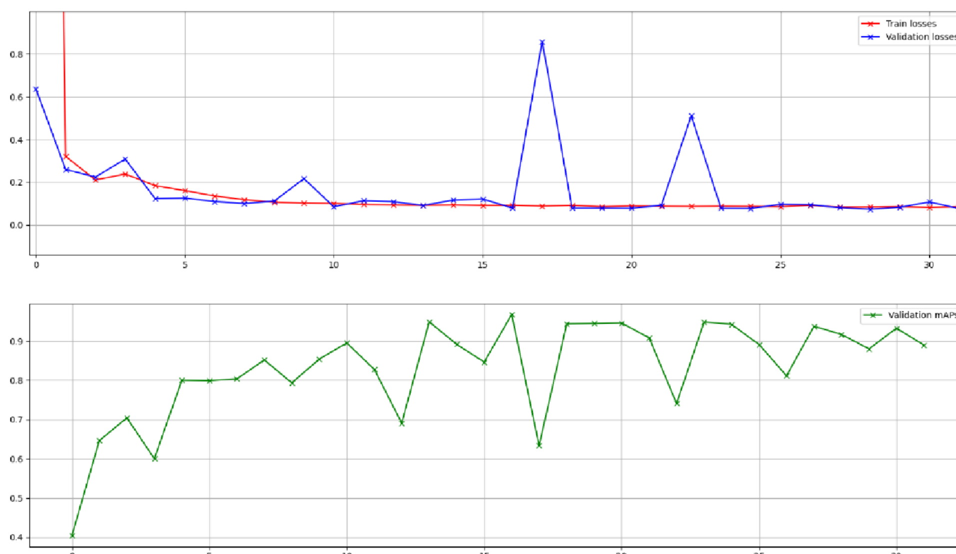


Obrázek 4.8 Predikce po krátkém testovacím učení. Bílý rámeček značí ground-truth, ostatní bounding boxy jsou predikce sítě. Confidence predikovaných bounding boxů jsou hodnoty před použitím funkce sigmoid.

5. UČENÍ A TESTOVÁNÍ

5.1 Učení

Pro učení byl využit počítač s grafickou kartou NVIDIA GeForce RTX 2080 Ti s pamětí 16 GB. Konfigurační konstanty byly nastaveny následovně. Learning rate = 0,001, weight decay = 0,0005 a batch size = 28 (dataset o 98000 vzorcích byl tímto rozdělen do 3500 vzorků). Vypočtení jednoho vzorku trvalo asi 0,4 vteřiny, což značí, že jedna učící epocha trvala necelých 25 minut. Po každé trénovací epoše proběhla validace, která kvůli nastavení batch size = 1 trvala přibližně 60 minut, na grafické kartě NVIDIA GeForce GTX 1050 trvala jedna validační epocha zhruba o 50 % více času (tj. 90 minut). Pro validaci i test bylo potřeba nastavit batch size na hodnotu 1 z toho důvodu, že například funkce NMS a mAP nebyly otestovány na větší hodnotu batch size a mohlo by dojít k promíchání bounding boxů mezi jednotlivými obrázky. Po 32 trénovacích a validačních epochách bylo učení ukončeno. Celkem tedy učení probíhalo bez mála 44,8 hodin. Výsledky během celého učení jsou zobrazeny na obr. 5.1.



Obrázek 5.1 Loss a mAP hodnoty v průběhu celého učení. Červená – trénovací loss, modrá – validační loss, zelená – validační mAP

5.2 Testování

Pro testování bylo nutné nejprve vybrat tu instanci, která má největší potenciál dosáhnout nejlepších výsledků. Ačkoliv je hodnota mAP v oblasti detekce objektu nejpoužívanější metrika, nesmí být opomenuta ani validační chyba sítě. Validační chyba sítě totiž může naznačit přeučování. Přeučování je pojem, který říká, že se model učí trénovací vzorky

nazpaměť. Pokud by takovýto model měl za úkol zpracovat data, která nikdy neviděl, nedokázal by výsledky určit tak přesně. Druhý výraz, podučení, naopak říká, že model se stále nedokázal dobře naučit vztah výstupů k vstupním datům. Jelikož je však důležitá hodnota mAP, budou zde zobrazeny výsledky dvou instancí (tabulka 5.1). Model A byla instance po 29 epochách učení, kdy validační loss dosáhl nejnižší hodnoty, a to 0,07358 a validační mAP byla 91,17 %. Model B určuje instanci po 16 epochách učení, kdy validační loss dosáhl hodnoty 0,0794 a rekordní validační hodnoty mAP50 - 96,73 %.

Tabulka 5.1 Výsledky implementace YOLOv3. Model A značí instanci, která v průběhu učení dosáhla nejnižší hodnoty chyby na validačním datasetu, model B označuje instanci, která v průběhu učení dosáhla nejvyšší hodnoty mAP na validačním datasetu

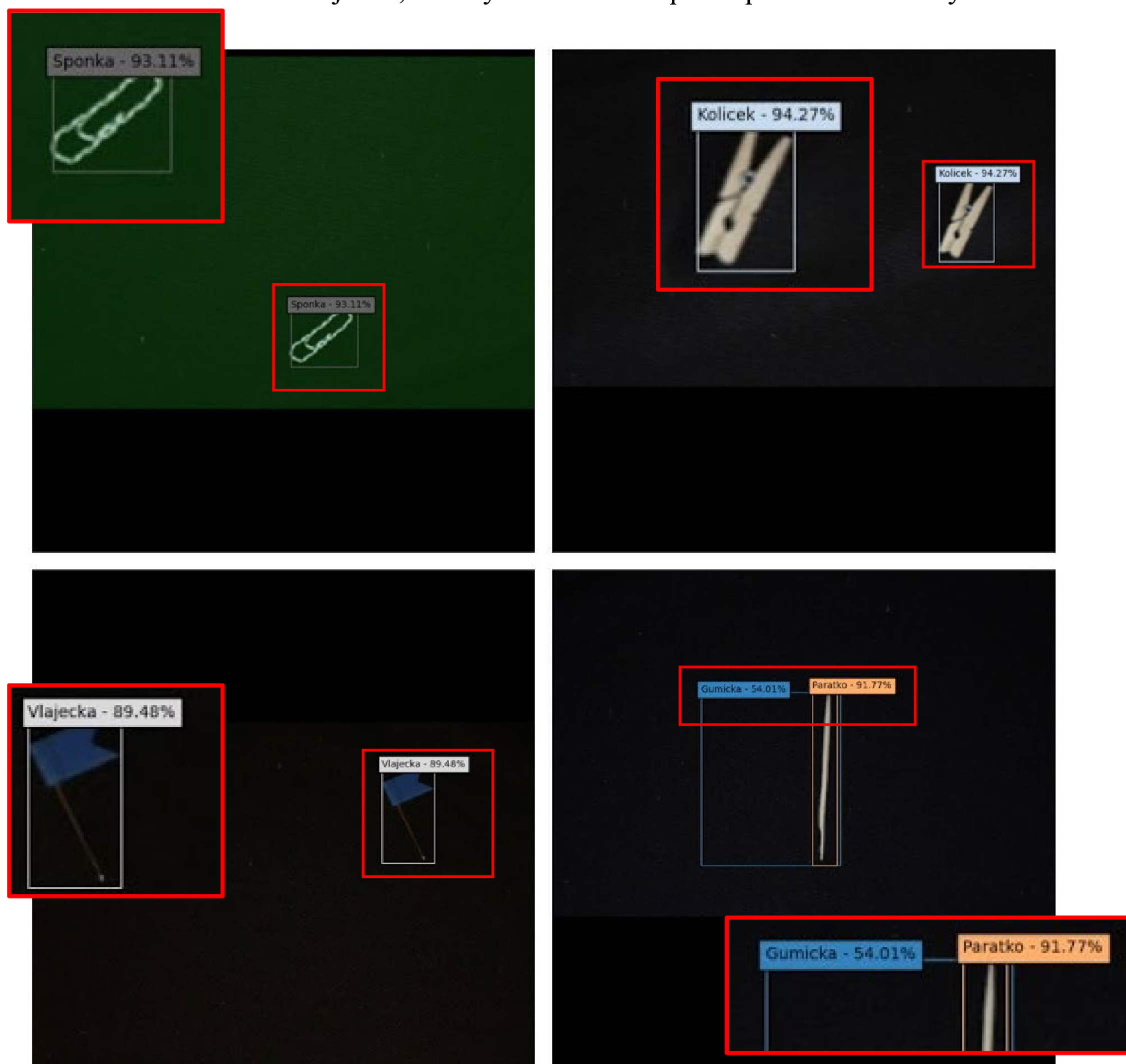
Model	loss [-]	mAP50 [%]
YOLOv3-A	0,0722	92,75
YOLOv3-B	0,0799	96,94

Hodnota mAP 96,94 % říká, že z asi třiceti dvou predikovaných bounding boxů, které mají větší confidence než 0,5, je pouze jeden nesprávný. Tyto hodnoty mAP jsou velmi dobré, dokonce lepší, než jakékoliv hodnoty mAP v tabulce 2.1 a na obrázcích 2.1, 2.2 a 2.3. Je však nutné podotknout, že tyto výsledky byly vypočteny na datasetu, který je focen na černém pozadí. Pokud by tato architektura byla použita pro fotografie s různorodějším obsahem, přesnost by nebyla tak vysoká. Dalším možným faktorem vysoké přesnosti v této implementaci mohlo být to, že 98 000 obrázků bylo rovnoměrně rozděleno mezi 7 tříd, a to znamená, že každý předmět byl architektuře předložen stejněkrát. Klasické datasety obsahují desítky klasifikačních tříd, které navíc nejsou v datasetu obsaženy rovnoměrně, tzn. že některé objekty architektura neumí detekovat zdaleka tak dobře, jako ty, které jsou v datasetu obsaženy vícekrát [30].

Na obr. 5.2 jsou vidět některé testovací fotografie. Na tomto obrázku vpravo dole je vidět jedna z fotografií, kde YOLOv3 určil nesprávný bounding box. Tento typ chyby byl primární zdroj špatných detekcí. Vždy, když nastala tato chyba, fotografie obsahovala správnou detekci s velmi vysokou confidence a další bounding box s nižší hodnotou confidence, obvykle jen něco málo přes 50 %. Pokud by se jednalo o architekturu, která by měla určit pouze jeden bounding box v každé fotografii, tak by se bounding box s nižší hodnotou pravděpodobnosti odfiltroval a s nadsázkou by se dalo říct, že na tomto datasetu by model dosáhl 100% úspěšnosti. Dalším možným řešením, méně invazivním, by bylo provádět NMS i mezi bounding boxy různých tříd, jelikož tyto false positive bounding boxy byly s true positive bounding boxy překryté velmi vysokým procentem.

Protože model detekuje předměty na fotografiích velmi dobře, dalo by se očekávat, že i confidence těchto bounding boxů budou vysoké, blížící se k 100 %, není tomu však úplně tak. Hodnoty confidence se obvykle pohybují kolem hodnoty 90 %. K vyřešení tohoto problému bylo nutné se znovu podívat na to, jak je tato hodnota zadefinována.

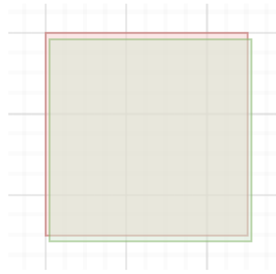
V kapitole 4.7 bylo řečeno, že hodnota confidence by měla odpovídat hodnotě IoU, kterou si model myslí, že má jeho predikovaný bounding box s ground truth bounding boxem. Tato metrika se však chová jinak, než by se mohlo na první pohled zdát. Aby se hodnota



Obrázek 5.2 Predikce architektury YOLOv3

IoU blížila ke 100 %, musely by být bounding boxy překryté opravdu téměř dokonale. Na obrázku 5.3 jsou zobrazeny bounding boxy s IoU 90 %, což potvrzuje, že pokud IoU má hodnotu alespoň zhruba 90 %, lze výsledek považovat za úspěšný.

Přesnost architektury byla tedy velmi uspokojivá, ale dalším důležitým parametrem, kvůli kterému byla architektura vybrána, byla rychlost. YOLOv3 je proslulá svou rychlostí (obr. 2.2), a jak bylo řečeno v kapitole 2.1, bylo potřeba, aby zpracování jedné fotografie trvalo maximálně 50 ms pro dosažení rychlosti 20 snímků za sekundu. Měření času probíhalo tak, že před vložením fotografie do sítě byly zapnuty stopky a jakmile se na této síti objevily výstupní hodnoty, stopky byly zastaveny. Protože se však časy



Obrázek 5.3 IoU 90 %

jednotlivých průběhů mohly lišit, bylo potřeba měření provést vícekrát a následně hodnoty zprůměrovat. Výsledný průměrný čas zpracování jedné fotografie, s rozlišením vstupu 320x320 px byl, se zaokrouhlením na celé ms, 13 ms. Tato hodnota odpovídá rychlosti 77 snímků za sekundu. Model je tedy vhodný i pro zpracování videa. Tento čas je dokonce nižší, než čas udaný v článku YOLOv3 (obr. 2.2). Zřejmě je to dáno tím, že v článku byl čas měřen včetně dalších funkcí, například NMS, nebo použitím lepší výpočetní techniky.

6. ZÁVĚR

Výsledky implementace architektury YOLOv3 byly velmi dobré, a proto lze říci, že zadání bylo splněno. Přesnost této architektury dosahovala velmi vysokých procent a rychlost byla dostatečná na to, aby se architektura dala použít pro real-time video. Tento model je tedy vhodný pro použití na běžícím dopravním pásu, jelikož použitý dataset obsahoval i rozmazané fotografie, a i na nich byly objekty správně detekovány. Bylo by však nutné zajistit, aby fotoaparát, který byl použit pro vytvoření trénovacího datasetu byl se stejným nastavením použit ke snímání objektů v ostrém provozu, jelikož různé fotoaparáty v různých prostředích fotí fotografie různých kvalit. Dataset použitý v této implementaci je velmi jednoduchý a stačilo by, kdyby dopravní pás obsahoval nějaké šmouhy, škrábance, nerovnoměrné osvětlení, nebo výrazné stíny objektů a výsledná přesnost by byla mnohem nižší.

Tato implementace architektury YOLOv3 je vhodná pro obecné použití, stačí jen splnit určitá kritéria toho, aby formát datasetu byl s touto implementací kompatibilní. Ačkoliv celá síť byla vytvořena tak, aby každá fotografie mohla obsahovat více předmětů k detekci, tato implementace postrádá načítání takovýchto obrázků a tato funkce by musela být doplněna. Důvodem je, že použitý dataset obsahuje fotografie, na kterých je vždy jen jeden předmět, a proto při implementaci nebyl uvažován formát anotace, který obsahuje více bounding boxů. Případná implementace této funkcionality by se musela týkat úpravy kódu zabývajícího se augmentací, anotací, úpravou fotografií a následně i samotného kódu, který načítá tyto vzorky do sítě.

Takto naimplementovaná síť je jednoduchá na použití, kód je přehledně rozdělen do několika částí tak, aby každá manipulace s daty byla intuitivně dohledatelná. Celý program byl vytvořen pro tento konkrétní dataset a vytvořené funkce byly používány pouze pro práci s tímto datasetem. Je proto možné, že se v kódu může vyskytnout několik „bugů“ (chyb). Nejpravděpodobněji se chyby mohou vyskytnout v částech, které se starají o tzv. „post-processing“, např. funkce NMS, nebo funkce pro výpočet mAP.

LITERATURA

[1] *neuroskills.com. Neuronal Firing* [online]. [cit. 2021-05-03]. Dostupné z: https://www.neuroskills.com/brain-injury/neuroplasticity/neuronal-firing/

[2] COPIACO, Abigail, Ritz C., Fasciani S. a Abdulaziz N. *Scalogram Neural Network Activations with Machine Learning for Domestov Multi-channel Audio Classification* [online]. University of Oslo, Norsko, University od Wollongong in Dubai, SAE, 2019 [cit. 2021-05-03]. Dostupné z: https://www.researchgate.net/publication/337977588_Scalogram_Neural_Network_Activations_with_Machine_Learning_for_Domestic_Multi-channel_Audio_Classification

[3] CHERRY, Kendra. *verywellminded.com. How Many Neurons Are In The Brain?* [online]. 2020 [cit. 2021-05-03]. Dostupné z: https://www.verywellmind.com/how-many-neurons-are-in-the-brain-2794889

[4] BISEN, Vikram Singh. *medium.com* [online]. *Where Is Artificial Intelligence Used: Areas Where AI Can Be Used,* 2019 [cit. 2021-05-03]. Dostupné z: https://medium.com/vsinghbisen/where-is-artificial-intelligence-used-areas-where-ai-can-be-used-14ba8c092e73

[5] DALEY, Sam. *builtin.com* [online]. *32 Examples Of AI In Healthcare That Will Make You Feel Better Abut The Future*, 2019 [cit. 2021-05-03]. Dostupné z: https://builtin.com/artificial-intelligence/artificial-intelligence-healthcare

[6] STATT, Nick. *theverge.com* [online]. *How Artificial Intelligence Will Revolutionize The Way Video Games Are Developed And Played*, 2019 [cit. 2021-05-03]. Dostupné z: https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning

[7] MOLTZAU, Alex. *medium.com* [online]. *Artificial Intelligence And Linguistics*, 2020 [cit. 2021-05-03]. Dostupné z: https://medium.datadriveninvestor.com/artificial-intelligence-and-linguistics-dc9eeee775dd

[8] STARR, Michelle. *sciencealert.com* [online]. *AI Simulates The Universe And Not Even Its Creators Know How It's So Accurate*, 2019 [cit. 2021-05-03]. Dostupné z: https://www.sciencealert.com/ai-simulates-the-universe-and-not-even-its-creators-know-how-it-s-so-accurate

[9] *paperswithcode.com* [online]. *Object Detection Models* [cit. 2021-05-03]. Dostupné z: https://paperswithcode.com/methods/category/object-detection-models

[10] GIRSHICK, Ross, Donahue J., Darrell T. a Malik J. *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)* [online]. UC Berkeley, California USA, 2014 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1311.2524.pdf

[11] UIJLINGS, R. Jasper, van de Sande K.E.A., Gevers T. a Smeulders A. W. M. *Selective Search for Object Recognition* [online]. University of Trento, Itálie, University of Amsterdam, Nizozemsko, 2012 [cit. 2021-05-03]. Dostupné z: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf

[12] GIRSHICK, Ross. *Fast R-CNN* [online]. Microsoft research, 2015 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1504.08083.pdf

[13] GANDHI, Rohith. *towardsdatascience.com* [online]. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO – Object Detection Alghorhitms*, 2018 [cit. 2021-05-03]. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[14] SHARMA, Pulkit. *analyticsvidhya.com* [online]. *A Step-by-Step Introduction to the Basics Object Detection Algorithms*, 2018 [cit. 2021-05-03]. Dostupné z: https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/

[15] REN, Shaoqing, He K., Girshick R. a Sun J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [online]. Univesity of Science and Technology of China, Hefei, Čína, Visual Computing Group, Microsoft research, Facebook AI Research, 2016 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1506.01497.pdf

[16] HE, Kaiming, Gkioxari G., Dollár P. a Girshick R. *Mask R-CNN* [online]. Facebook AI research (FAIR), 2017 [cit. 2021-05-03]. Dostupné z: https://openaccess.thecvf.com/content_ICCV_2017/papers/He_Mask_R-CNN_ICCV_2017_paper.pdf

[17] KELLY, Adam. *immersivelimit.com* [online]. *Mask R-CNN with TensorFlow 2 on Windows 10,* 2020 [cit. 2021-05-03]. Dostupné z: https://www.immersivelimit.com/tutorials/mask-rcnn-for-windows-10-tensorflow-2-cuda-101

[18] REDMON, Joseph, Divvala S., Girshick R. a Farhadi A. *You Only Look Once: Unified, Real-Time Object Detection* [online]. University of Washington, Washington, USA, Allen Institute for AI, Facebook AI Research, 2016 [cit. 2021-05-03]. Dostupné z: https://pjreddie.com/media/files/papers/yolo_1.pdf

[19] *araintelligence.com* [online]. *YOLOv1: You Only Look Once* [cit. 2021-05-03]. Dostupné z: https://araintelligence.com/blogs/deep-learning/object-detection/yolo_v1/

[20] REDMON, Joseph a Farhadi A. *YOLO9000: Better, Faster, Stronger* [online]. University of Washington, Washington, USA, Allen Institute for AI, XNOR.ai., 2017 [cit. 2021-05-03]. Dostupné z: https://openaccess.thecvf.com/content_cvpr_2017/papers/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.pdf

[21] LIU, Wei, Anguelov D., Erhan D., Szegedy C., Reed S., Fu C. a Berg A. C. *SSD: Single Shot MultiBox Detector* [online]. UNC Chapel Hill, Zoox Inc., Google Inc., Ann-Arbor University of Michigan, Michigan, USA, 2016 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1512.02325.pdf

[22] REDMON, Joseph a Farhadi A. *YOLOv3: An Incremental Improvement* [online]. University of Washington, Washington, USA, 2018 [cit. 2021-05-03]. Dostupné z: https://pjreddie.com/media/files/papers/YOLOv3.pdf

[23] KAMAL, Amro. *medium.com* [online]. *YOLO, YOLOv2 and YOLOv3: All You want to know*, 2019 [cit. 2021-05-03]. Dostupné z: https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899

[24] LIN, Tsung-Yi, Goyal P., Girshick R., He K. a Dollár P. *Focal Loss for Dense Object Detection* [online]. Facebook AI Research (FAIR), 2018 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1708.02002.pdf

[25] *argcis.com* [online]. *How RetinaNet works?* [cit. 2021-05-03]. Dostupné z: https://developers.arcgis.com/python/guide/how-retinanet-works/

[26] ANWLA, Praveen Kumar. *analyticsvidhya.com* [online]. *A Beginner's Guide to Focal Loss in Object Detection!,* 2020 [cit. 2021-05-03]. Dostupné z: https://www.analyticsvidhya.com/blog/2020/08/a-beginners-guide-to-focal-loss-in-object-detection/

[27] JAEGER, Paul F., Kohl S. A. A., Bickelhaupt S., Isensee F., Kuder T. A., Schlemmer H. a Maier-Hein K. H. *Retina U-Net: Embarrassingly Simple Exploitation of Segmentation Supervised for Medical Object Detection* [online]. Division of Medical Immage Computing, Department of Radiology, Medical Physics in Radiology, German Cancer Research Center, Heidelberg, Německo, 2018 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1811.08661.pdf

[28] LI, Yixing, Ren F. *Light-Weight RetinaNet for Object Detection* [online]. School of Computing, Informatics, and Decision Systems Engineering Arizona State University, Arizona, USA, 2019 [cit. 2021-05-03]. Dostupné z: https://arxiv.org/pdf/1905.10011.pdf

[29] RIGGIO, Christopher. *towardsdatascience.com* [online]. *What's the deal with Accuracy, Precision, Recall and F1?*, 2019 [cit. 2021-05-03]. Dostupné z: https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021

[30] DUBEY, Vijay. *medium.com* [online]. *Popular Object Detection datasets – Analysis and Statistics*, 2020 [cit. 2021-05-03]. Dostupné z: https://medium.com/@vijayshankerdubey550/popular-object-detection-datasets-analysis-and-statistics-66acdacc3aa9

[31] DUTTA, Abhishek, Gupta A. a Zisserman A., *robots.ox.ac.uk* [online]. *VGG Image Annotator (VIA)* [cit. 2021-05-03]. Dostupné z: https://www.robots.ox.ac.uk/~vgg/software/via/

[32] COSTA, Claire D. *towardsdatascience.com* [online]. *Best Python Libraries for Machine Learning and Deep Learning*, 2020 [cit. 2021-05-03]. Dostupné z: https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

[33] KATHURIA, Ayoosh. *towardsdatascience.com* [online]. *What's new in YOLOv3?*, 2018 [cit. 2021-05-03]. Dostupné z: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[34] AMJOUD, Ayoub Benali a Amrouch M. *springer.com* [online]. *Convolutional Neural Networks Backbones for Object Detection*, 2020 [cit. 2021-05-03]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-030-51935-3_30

[35] REDMON, Joseph. *github.com* [online]. *yolov3.cfg*, 2018 [cit. 2021-05-03]. Dostupné z: https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg

[36] *pytorch.org* [online]. *Cuda Semantics* [cit. 2021-05-03]. Dostupné z: https://pytorch.org/docs/stable/notes/cuda.html

[37] *pytorch.org* [online]. *Torch.utils.data* [cit. 2021-05-03]. Dostupné z: https://pytorch.org/docs/stable/data.html

[38] SANDEEP, Ananth. *medium.com* [online]. *Object Detection -IOU-Intersection Over Union*, 2019 [cit. 2021-05-03]. Dostupné z: https://medium.com/@nagsan16/object-detection-iou-intersection-over-union-73070cb11f6e

[39] LODAYA, Tejas. *github.com* [online]. *README.md: YOLO*, 2018 [cit. 2021-05-03]. Dostupné z: https://github.com/tejaslodaya/car-detection-yolo/blob/master/README.md

[40] ALMOG, Uri. *towardsdatascience.com* [online]. *YOLO V3 Explained*, 2020 [cit. 2021-05-03]. Dostupné z: https://towardsdatascience.com/yolo-v3-explained-ff5b850390f

SEZNAM PŘÍLOH

PŘÍLOHA A - ZDROJOVÝ KÓD ULOŽENÝ NA PŘILOŽENÉM CD („BP_YOLOV3“)............ 47

Příloha A - Zdrojový kód uložený na přiloženém CD („BP_YOLOv3“)

/……………………………………………………Kořenový adresář přiloženého CD

YOLOv3.zip………………………………Komprimovaný soubor se zdrojovými kódy

YOLOv3……………………………………………Složka se zdrojovými kódy

main.py…………………………........………………Spouštěcí python soubor

TRAIN.py……..……………………...Python soubor s funkcemi train a play

DATASET.py…………………………..Python soubor s třídou ImageLoader

YoloV3.py…………….Python soubor s architekturou YOLOv3 a loss funkcí

IOU.py………...…………….Python soubor s funkcí Intersection over Union

NMS.py……...………………..Python soubor s funkcí Non-max suppression

MAP.py….......……...…...Python soubor s funkcí pro výpočet hodnoty mAP

TARGET_CONVERSION.py......Python soubor s funkcemi pro převod mezi targety a bounding boxy

SHOW_IMAGE.py………………....Python soubor s funkcí pro zobrazování výsledných obrázků

IMG_PREPROCESS.py……...……..Python soubor s funkcemi pro přípravu datasetových obrázků