VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2020 **Bc. Jan Křístek**

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
katedra informatiky

Detekce reálných objektů podle syntetických předloh

Object Detection Based on Synthetic Images

2020 **Bc. Jan Křístek**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Křístek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce reálných objektů podle syntetických předloh
Object Detection Based on Synthetic Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je navrhnout a otestovat aplikaci pro vyhledávání objektů v obraze na základě syntetických předloh (vyrenderovaný obraz hledaného objektu). Výsledná implementace bude schopna odhadnout polohu objektu v obraze a určit hrubý odhad jeho rotace vůči pozorovateli (např. rozlišit mezi šesti směry pohledů na objekt). Předpokládá se využití existujícího frameworku pro vytváření neuronových sítí (TensorFlow nebo Darknet) a knihovny OpenCV.

1. Nastudujte zvolený framework pro vytváření neuronových sítí.
2. Navrhněte nebo využijte existující architekturu sítě umožňující rozpoznat zvolený objekt v obraze a zároveň určit jeho umístění.
3. Ověřte funkčnost sítě na vhodném datasetu (např. pomocí mAP).
4. Problém rozšiřte o možnost odhadu rotace objektu a opět proveďte vyhodnocení úspěšnosti.
5. Postup a dosažené výsledky pečlivě popište v textové části práce.

Seznam doporučené odborné literatury:

[1] YU, Jiahui, et al. Unitbox: An advanced object detection network. In: Proceedings of the 24th ACM international conference on Multimedia. ACM, 2016. p. 516-520.
[2] ZHOU, Xinyu, et al. EAST: an efficient and accurate scene text detector. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2017. p. 5551-5560.
[3] Joseph R. Darknet: Open source neural networks in c. Pjreddie.com. 2016.

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

Vedoucí diplomové práce: **Ing. Tomáš Fabián, Ph.D.**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě-Porubě dne 13. 7. 2020

.....
Podpis

Poděkování

Rád bych poděkoval vedoucímu diplomové práce Ing. Tomáši Fabiánovi, Ph.D., za odbornou pomoc a konzultaci při vytváření této práce.

Abstrakt

Cílem této diplomové práce je navrhnout a otestovat aplikaci pro vyhledávání objektů v obraze na základě syntetických předloh. Výsledná implementace bude schopna odhadnout polohu objektu v obraze a určit hrubý odhad jeho rotace vůči pozorovateli. Provedl jsem výzkum a testoval jsem několik neuronových sítí pro řešení problému, a sice prvně se zadaným Unitboxem, následně s YOLem v Darknetové implementaci a v závěru s TensorFlow implementací a vlastní implementací. Podařilo se mi vytvořit sítě, které splnily zadání.

Klíčová slova

syntetická předloha, neuronové sítě, YOLO

Abstract

The purpose of this diploma thesis is to develop and test software application for finding objects in images based on synthetic pattern. Resulting application will be able to guess position of object in the image and guess harsh estimate of its rotation for user. I made research a tested several neural networks for solving this problem. I stared with Unitbox and continued with Darknet implementation of YOLO and finally with TensorFlow YOLO implementation and own implementation. I managed to create network that satisfies the assignment.

Key words

synthetic pattern, neural network, YOLO

Seznam použitých symbolů a zkratek

API	rozhraní pro programování aplikací (Application Programming Interface)
CAD	počítačem podporovaný design (Computer Aided Design)
CAT	počítačová axiální tomografie (Computerized Axial Tomography)
CNN	konvoluční neuronová síť (Convolutional Neural Network)
HDR	s vysokým dynamickým rozsahem (High Dynamic Range)
IOU	průnik nad sjednocením (Intersection over Union)
JIT	právě včas metoda překladu zdrojového kódu (Just In Time)
MSE	průměrná chyba nadruhou (Mean Squared Error)
QR	labyrintový kód (Quick Response Code)
RGB	barevný model červená-zelená-modrá
RPG	hra s rolemi (Role Playing Game)
VGG	vizuální geometrická skupina (Visual Geometry Group)
YOLO	podíváš se jednou (You Only Look Once)

Seznam tabulek

Seznam obrázků

Obsah

1. Úvod

Problém, kterým se tato práce zabývá, spočívá v nalezení specifického objektu v obraze a následném nalezení úhlu, z něhož objekt vidíme. Základní kroky se dají popsat v následujících bodech:

- načtení obrázků,
- identifikace objektu a určení jeho umístění,
- nalezení úhlu natočení objektu,
- zobrazení výsledku.

Dva hlavní body, které musí výsledné řešení splnit, je nalezení objektu s jeho pozicí v obraze a následně natočení objektu. Tyto úkoly může zpracovat jediná síť najednou, nebo se dá použít pro implementaci dvojice sítí. Nalezení konkurenčních řešení nebylo jednoduché hlavně kvůli použití slova úhel, ale pro popis problému se více hodí slovo póza. Zpracování úhlu je jednodušší z pohledu složitosti úlohy. Uplatnění řešení tohoto problému by se dalo využít ve výrobních robotech, automobilovém průmyslu, počítačových hrách, virtuální a rozšířené realitě. Cílem je navrhnout aplikaci, neuronovou síť, která nalezne požadovaný objekt v obraze a podle výstupu určí úhel natočení hledaného objektu. Reálné testovací a syntetické trénovací datasety jsou přidány k zadání. Výsledkem bude implementace a následná analýza dosažených výsledků.

Obsahem diplomové práci je popis aplikace, informace o knihovnách použitých pro její vývoj, základní informace o neuronových sítích, popis syntetických obrázků a jejich rozdíl ve vztahu k reálným obrázkům. Popis aplikace je rozdělen na popis aplikace a výsledků. Tyto body jsou obsaženy v následujících kapitolách.

V kapitole 2 je popsán přírodní neuron a jeho fungování a následně počítačový neuron neboli perceptron. V další části jsou popsány neuronové sítě a to, jak fungují doplněné o známé příklady konvolučních neuronových sítí. Kapitolu uzavírají datasety, které jsou důležitou součástí práce s neuronovými sítěmi. Jsou uvedeny kategorie tříd a existující známé datasety uzavřené popisem trénovacího a testovacího datasetu. Na konci kapitoly budou představeny syntetické obrazy a reálné obrazy a příklady použití těchto synteticky generovaných obrázků při trénování neuronových sítí. V kapitole 3 knihovna OpenCV a Tensorflow jsou uvedeny informace o nejnovější verzi TensorFlow 2.0 a to, k jakým změnám v knihovně došlo. Kapitolu uzavírá popis, jak vypadají označovací soubory k obrázkům z datasetů a jaké programy a přístupy se dají použít pro vygenerování těchto souborů. V kapitole 4 je uveden popis samotné aplikace. Bude zde popis použité Darknetové YOLO implementace, která byla používána do prosince 2019. Následuje popis vlastní implementace v TensorFlow 2.0 a Pythonu, a to v programu na provedení celého natrénování se zobrazením výsledků a načtení natrénovaných vah a zobrazení výsledků. V poslední části kapitoly je popsána TensorFlow 2.0 implementace konvoluční neuronové sítě YOLO a úpravy, které jsem v ní provedl. V kapitole 5 je popis výsledků sítí pro trénování úhlů a výsledků z Darknetové implementace. V poslední části je popis výsledků natrénované vlastní implementace a upraveného YOla a jejich zhodnocení. Poslední částí je závěr, ve kterém se nachází závěrečné zhodnocení.

2. Počítačové vidění

V této kapitole jsou popsány neuron, perceptron a neuronové sítě. Kapitolu uzavírají datasety a syntetické obrazy.

Počítačové vidění může najít uplatnění v oblasti:

- Zabezpečení: detekce nedovoleného vniknutí cizí osoby do objektu, rozpoznání tváří a státní rozpoznávací značky. Systémy pro detekci tváří jsou jedním z důležitých prvků nového čínského sociálního kreditového systému.
- Automobilový průmysl: detekce tváře řidiče pro zjištění údajů o koncentraci řidiče, nebo jeho hrubém odhadu aktuálního zdravotního stavu pro systém automatického zastavení. S rostoucím vývojem v oblasti samořiditelných aut je další důležitým zaměřením v této oblasti detekce okolí automobilu kvůli detekci překážek, chodců a dalších aut. Detekce provozu je využívána velkým množstvím navigačních aplikací.
- Zdravotnictví: systém na výpočet ztracené krve během lékařského zákroku.
- Zemědělství: systémy, které umožní farmáři zvýšit efektivitu a výdělky.
- Obchod: příkladem jsou prodejny Amazon GO, kde zákazník nemusí čekat ve frontě. Před vstupem si zákazník aktivuje mobilní aplikaci, ta se napojí na systém obchodního řetězce. Uvnitř obchodu kamery snímají police a lidi, kteří si z dané police vezmou zboží. Po nákupu zákazník odejde z obchodu a dorazí mu online vyúčtování.

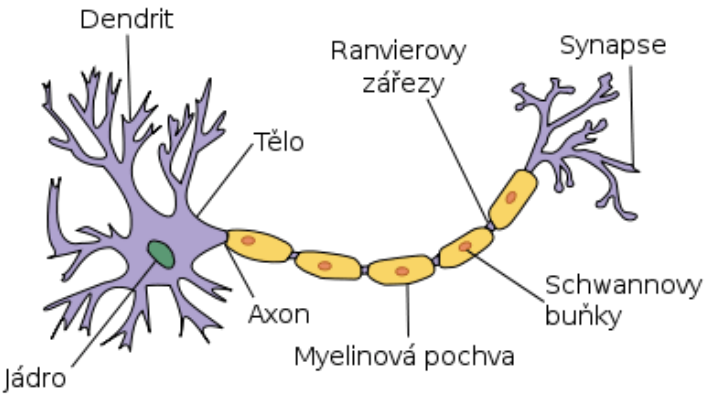
2.1 Neuron

V této části bych chtěl popsat neuron, a to z hlediska biologického i programového. Cílem je nastínění problematiky pro pochopení tématu. Neuron tvoří základní stavební jednotku neuronové sítě, proto by bylo dobré uvést základní informace o tomto důležitém základním kameni z biologického i programového pohledu.

Z pohledu biologie je neuron nervová buňka, která se vyskytuje v mozku. Jedná se o základní stavební jednotku nervové tkáně. Jedná se o specializované buňky, které mají za úkol vedení, zpracování a odpovídání na signály. Přenášejí informace z mozku do jednotlivých částí těla a opačně a tímto procesem umožňují tělu reagovat na vnější a vnitřní podněty. Člověk má v těle 15 až 25 miliard neuronů a 300 miliard kontaktních ploch neboli synapsí. Neuron má následující části:

- tělo,
- dendrity,
- neurity,
- iniciální část je místem odstupu dlouhého vlákna neuritu a také místem vzniku dalšího vzruchu,
- synaptická část je převodní část napojující se na další nervové buňky.

Tělo neuronu tvoří největší část neuronu. Tělo je ohraničeno membránou, obsahuje receptory a iontové kanály. Tato struktura podmiňuje šíření signálu. Uvnitř neuronu se nachází jádro, mitochondrie a Nisslova substance.



Obrázek 1: Ukázka stavby multipolárního neuronu [1]

Dendrity jsou vstupním místem, odkud se signál dostane do neuronu. Většinou jsou tyto výběžky krátké, bohatě větvené, rozšířené do dendritických trnů, které slouží k modulaci postsynaptického potenciálu při jeho přechodu ze synapse na dendrit. Jsou bohaté na chemicky řízené iontové kanály.

Neurity jsou dlouhé výběžky, které vedou výsledný signál z neuronu do dalšího neuronu. Obsahují ribozomy, malé množství mitochondrií a neurotubuly. Jsou bohaté na chemicky řízené iontové kanály. Neurity se skládají z axonu a synapse.

Na začátku axonu probíhá sumace informací, které přišly z různých dendritů. Pokud sčítání překročí práh významnosti, Axon začne šířit signál, který funguje na elektrickém principu. Signál doputuje až do zakončení neuronu a vyvolá synapsi: chemický děj, kterým se signál přenese z jednoho neuronu na druhý. Zde hrají důležitou roli neurotransmitery (chemické látky roznášející informace po nervovém systému a působící na psychiku člověka). Mezi nejznámější neurotransmitery patří např. dopamin, serotonin, noradrenalin.

Synapse se dá zjednodušeně představit na principu: na konci dendrického výběžku jsou váčky plné neurotransmiterů. Na začátku druhého neuronu jsou receptory. Mezi neurony je mezera, kterou projde signál. Příchod signálu způsobí, že obsah synaptických vezikul se uvolní a spadne do synaptické štěrbiny. Neurotransmitery jsou jako klíče, které se snaží zapadnout do zámku.

V lidském těle se nacházejí neurony různých velikostí, tvarů a funkcí. Nejzákladnější rozdělení neuronů je na senzitivní, motorické neurony a interneurony. Senzitivní neurony přivádějí informaci ze smyslových orgánů do mozku, kde jsou pak pomocí interneuronů propojeny na motorické neurony. Díky motorickým neuronům se informace dostane až do svalu.

Výběžky rozlišujeme na několik typů neuronů:

- bipolární neurony,
- unipolární neurony,
- pseudounipolární neurony,
- multipolární neurony.

Unipolární neurony mají pouze jeden výběžek, který se nazývá axon. Dendrit tvoří vzruch na základě svého podráždění přijatou informací. Unipolární jsou smyslové neurony, které mají za úkol přijímat informace. Příkladem jsou primární smyslové buňky, tyčinky a čípky sítnice.

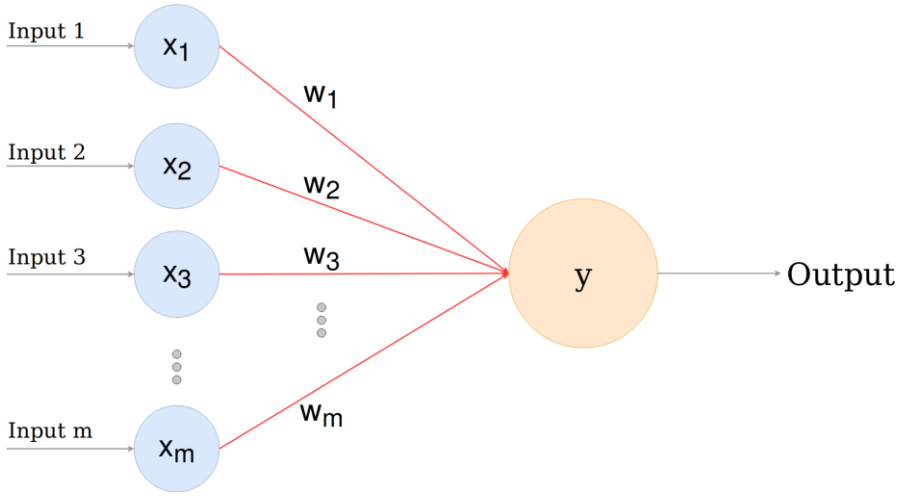
Bipolární neurony obsahují jeden neurit a jeden dendrit, které obvykle odstupují na opačných pólech buňky. Příkladem je druhý neuron čichové buňky nebo zrakové dráhy.

Pseudounipolární neuron je méně častý typ neuronu. Dendrit a axon splývají v jediný výběžek, který se nazývá dendraxon. Pseudounipolární neurony jsou například používají jako ganglia mozkových nervů.

Multipolární neurony jsou nejpočetnějšími typy neuronů. Buňka má hvězdicovitý tvar, protože z jejího těla vystupuje několik dendritů a jeden axon.

2.2 Perceptron

Programový neuron se nazývá perceptron, což je matematický model biologického neuronu. Jedná se o ústřední prvek neuronové sítě. První umělé neurony byly vytvořeny Warrenem McCullochem a Walterem Pittsem v roce 1943. Tyto neurony fungovaly tak, že jejich výstup byl 0 nebo 1 v závislosti na tom, jestli suma vstupních signálů překročila prahovou hranici, nebo ne.



Obrázek 2: Princip perceptronu [2]

U perceptronu počítáme 1 až N vstupů podle druhu zadané úlohy a indikovaný výstup. Vstupy jsou podněty z vnějšího okolí, nebo výstupy z jiných neuronů u vícevrstvých sítí. Každý vstup (X) je rozšířen o váhu (w) tohoto vstupu, která určuje jakou důležitost tento vstup má. Každým i-tým vstupem získá neuron informaciv daném časovém okamžiku jako reálné číslo. Jednotlivé vstupní spoje mají danou důležitost pomocí synaptických vah. Vektor vah musí být kolmý na rozhodovací hranici. Možností nastavení vah je nekonečně mnoho. Další veličinou neuronu je práh, jenž je považován za speciální případ váhy spoje vedoucího do fiktivního neuronu s trvalým výstupem −1. Vzorec je:

$$\vartheta = \sum_{i=0}^{N} w_i \cdot x_i = \mathrm{w} \cdot \mathrm{x}$$

- w je váha
- x je vstup

Samotný perceptron pak obsahuje prahovou funkci, při jejím překonání je perceptron aktivován a indikuje signál na svém výstupu ve formě přenosové funkce. U neuronových sítí se používá velké množství přenosových funkcí. V procesu testování se dospělo k využití zejména funkce jednotkového

skoku, funkce signum, sigmoidální funkce a funkce lineární. Výběr vhodné přenosové funkce neuronu má vliv na konvergenci výpočtu naučení neuronové sítě.

Perceptron si lze zjednodušeně představit jako binární klasifikátor, který mapuje vektor vstupů na výstupní hodnoty.

$$f(x) = \begin{cases} 1 \; pro \; w \cdot x + b > 0 \\ 0 \; pro \; w \cdot x + b \leq 0 \end{cases}$$

- w je vektor vah
- b je konstanta.

Přenosových funkcí je velké množství různých druhů. Příkladem je skoková přenosová funkce, která vrací pro vstup menší než daná mez nulu, pro větší vrací jedna. Dalším možným tvarem je lineární a sigmoidová přenosová funkce, která je ve tvaru:

$$\vartheta = \frac{1}{1 + \exp\left(-\alpha \cdot \vartheta\right)}$$

Jedná se o monotónně rostoucí funkci mezi dvěma asymptotickými hodnotami. Hodnotami jsou nejčastěji 0 a 1. Výhodou sigmoidální funkce je, že má ve všech bodech spojité derivace.

Cílem učení neuronové sítě je nastavit váhy modelu neuronové sítě tak, aby vytvářely správnou odezvu výstupního signálu na daný signál vstupu. Po procesu naučení neuronové sítě se lze na síť jako na připravenou pro využití k nasazení ve zvolených aplikačních rovinách neuronových sítí. Učení lze rozdělit do dvou typů:

- Učení bez učitele: Váhy sítě se nastavují tak, aby výstup byl konzistentní, tedy aby síť poskytovala stejnou odezvu na budicí signál při stejných, nebo podobných vektorech vstupu.
- Učení s učitelem: Neuronová síť se snaží srovnáváním aktuálního výstupu s požadovaným výstupem přenastavit váhy sítě tak, aby se na daný konkrétní vstup snížil rozdíl mezi skutečným a požadovaným výstupem.

Nejzákladnější úloha, kterou lze řešit perceptronem, je klasifikace perceptronem. Může se jednat například o klasifikaci objektu v obrazu do dvou skupin, a to na kočky a psy, jablka a hrušky, maliny a ostružiny a nekonečně mnoho dalších možností.

V minulých letech byl kladen důraz na metody bez strojového učení, mezi něž náleží například scale invariant feature transform (SIFT) a Histogram of oriented gradients (HOG) – byly užívány pro lineární klasifikaci a detekci vlastností, na jejichž základě došlo k predikci umístění objektu. Pozdější algoritmy se zaměřily na třídně nezávislé objektové návrhy umístění objektu s použitím segmentace a klasifikace použitím ručně vytvořených vlastností.

2.3 Neuronové sítě

Dnešní metody jsou založeny na strojovém učení a neuronových sítích a došly k state of the art výsledkům při klasifikaci obrázků, segmentaci a detekci objektů. Vývoj neuronových sítí je zpomalen absencí kvalitních a obsáhlých datasetů. Z neuronových sítí se jako nejlepší odnoží ukázaly být konvoluční neuronové sítě – (CNN), které dokázaly splnit předpoklad pro robustní extraktor vlastností. Konvoluční neuronové sítě začaly jako klasifikátory obrázků a postupem času se začaly používat pro detekci objektů. Neuronové sítě se využívají pro řešení problémů jako:

Klasifikace obrázků je základní problém, který neuronové sítě řeší. Klasifikace obrázků bere obrázek a předpovídá objekt v obrázku. Příkladem je klasifikátor, který rozpozná psa nebo kočku v obrázku. Pokud je v obrázku více objektů, je potřeba použít multi-label klasifikátor.

Lokalizace objektu v obrázku je další důležitý problém. Nejčastěji bývá umístění objektu v predikčním obrázku vykresleno jako obdélník, v případě satelitních a leteckých snímků se využívají otáčené obdélníky.

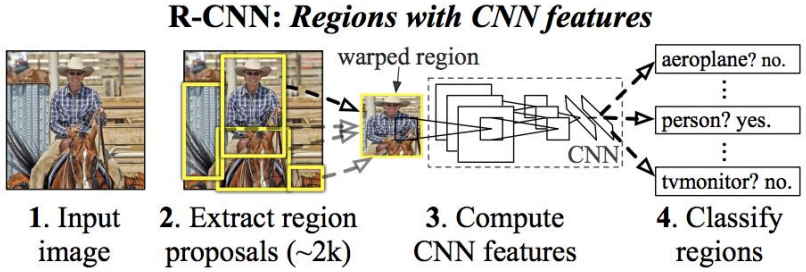
Kombinace klasifikace objektu a jeho lokalizace se dohromady nazývají detekce objektu. V neuronových sítích nejčastěji dochází k lokalizaci objektu a jeho klasifikaci. Výsledkem je výstupní obrázek s obdélníky, které označují umístění objektu. U každého obdélníku se nachází značka třídy s nejvyšší pravděpodobností a dané procento. Výstupem je nejčastěji tento datový formát:

- číslo třídy,
- levá horní x pozice,
- levá horní y pozice,
- šířka,
- výška.

Tyto hodnoty jsou celá čísla, mohou nabývat intervalu od 0 teoreticky do nekonečna. V praxi jsou omezena počtem tříd v datasetu a vhodnou velikostí obrázků. Existuje i modifikace tohoto výstupu, kdy levý horní x a y jsou nahrazeny centrálním x a y. Šířka a výška obdélníku jsou v tomto případě poloviční. Následující popisky konvolučních neuronových síťí jsou založeny na [6].

Mezi anchor free neuronové sítě, které se hodí pro semestrální projekt, patří například:

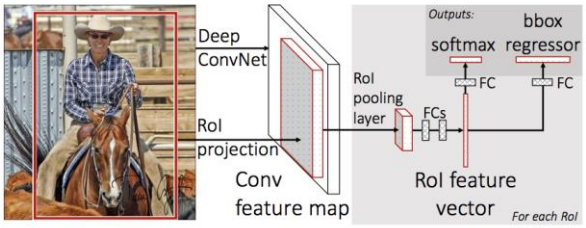
R-CNN: Byla první konvoluční neuronová síť, která umožnila velký vývoj tohoto odvětví. Předtím se používaly jiné konvoluční neuronové sítě, které byly velmi pomalé. Tento malý výkon byl dán použitím Sliding Window algoritmu. R-CNN byla první neuronovou sítí, která místo Sliding Window používala Selective Search. Použití tohoto algoritmu pro vyhledávání možných umístění objektů vedlo k redukci počtu návrhových čtverců, které byly přeneseny do klasifikátoru. Maximum vyhledaných míst je 2000. Tato umístění jsou zabalena a odeslána do neuronové sítě, která vyrobí 4096 dimenzionální vektor. Síť se zaměřuje na vlastnosti v obrázku a vytvoří z těchto vlastností vrstvu, tu dále pošle do SVM (Support Vector Machine), který zjišťuje přítomnost objektu v daném Object Proposal.



Obrázek 3: Princip fungování R-CNN [3]

Spacial pyramid pooling (SPP-net) vzniklo, protože R-CNN neměla dostatečnou rychlost. Vypočítáme CNN reprezentaci pro celý obrázek a tu použijeme pro každý Object Proposal ze Selective Search.

Fast R-CNN je založená na C-NN, rozdíl spočívá v tom, že rovnou vstupní obrázek pošleme do neuronové sítě, jež vytvoří mapu vlastností a tu zpracuje Selective Search do Object Proposals. Toto řešení vedlo k podstatnému zrychlení algoritmu.



Obrázek 4: Princip fungování Fast R-CNN [4]

Faster R-CNN na rozdíl od dvou výše uvedených nepoužívá Selective Search pro nalezení Object Proposals a tuto úlohu přenechává neuronové síti. Pro zrychlení se používá dopředu trénovaná síť nad ImageNet datasetem. Jako u Fast R-CNN se vstupní obrázek pošle do neuronové sítě pro vygenerování mapy vlastností a tato mapa se pošle do druhé neuronové sítě (RPN – Region proposal network), která vygeneruje Object Proposals.

Region Proposal Network (RPN) je konvoluční neuronová síť, která se používá ve Faster R-CNN a má za úkol vytvořit seznam možných umístění objektu. Prvním krokem je vygenerování konvoluční mapy vlastností přes konvoluční neuronovou síť pro zadaný vstupní obrázek,pro nějž jsou důležité hodnoty scale a aspect-ratio. Vývojáři jim dávají hodnotu 3, pro každý pixel tak může být maximálně K = 9 možných umístění obrazu na pixelu. Používá se Sliding Window a uprostřed každého Sliding Window okna je kotva, která může být celkem W × H × K. W = šířka a H = výška. Algoritmus každé kotvě přiřadí značku podle dvou faktorů:

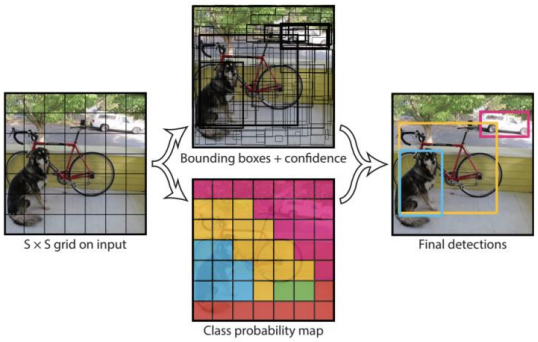
1. kotva s nejvyšším Intersection-over-union overlap.
2. kotva s Intersection-over-union overlap vyšší než 0,7.

Další část RPN obsahuje dvě důležité části classifier a regressor. Classifier určuje s jakou pravděpodobností má návrh cílový objekt a regressor souřadnice návrhu. Provádí se zde extrakce dat z mapy vlastností a určí se clasiffier a regressor. Neuronovou síť je potřeba trénovat, určitě tak můžeme použít ztrátovou funkci:

$$L(\{p_i\},\{t_i\}) = \left(\frac{1}{N_{cls}}\right) \cdot \sum L_{cls} \cdot (p_i, {p_i}^*) + \left(\frac{\lambda}{\mathrm{N_{reg}}}\right) \cdot \sum {p_i}^* \, L_{reg}(t_i, {t_i}^*)$$

i = index kotvy, $\mathrm{p_i}$ = pravděpodobnost, že je objektem, $\mathrm{t_i}$ = vector koordinátů pro ohraničení, * = reprezentuje hodnotu pravdivostní tabulky, L = reprezentuje ztrátu. p^* zajišťuje, že když se identifikuje objekt, tak se započítá jen regrese. $\mathrm{N_{cls}}$ a $\mathrm{N_{reg}}$ = normalizace. λ je obvykle 10.

YOLO je oproti výše zmíněným charakteristickým použitím jediné neuronové sítě, která pro celý obrázek zjistí možná umístění objektu. Proces začíná rozdělením obrázku na s × s matici, v každé mřížce je n ohraničení, pro každé ohraničení síť vypočítá třídu pravděpodobnosti a posun hodnot pro ohraničení. V současnosti je nejvíce používaná verze 3. YOLO, jež je k dispozici díky množství knihoven a frameworků. Implementace je dostupná například ve frameworku Darknet a knihovnách TensorFlow, PyTorch, TensorRT a dalších.



Obrázek 5: Princip fungování YOLO [5]

Single Shot Detector (SSD) používá CNN pro detekci vlastností obrázku. Poté spustí 3 × 3 matici nad mapou vlastností a zjišťujeme Object Proposals.

Spatial Pyramid Poolong (SPP-net) je CNN, která vznikla jako vylepšení R-CNN. Rozdíl je v tom, že SPP-net počítá reprezentaci obrázku jen jednou a toto je použito sítí pro každé možné umístění generované algoritmem Selective Search.

UnitBox je založený na principu nezávislosti 4 hodnot ohraničujícího rámce, tento princip nebývá u ostatních konvolučních neuronových sítí dodržen a je vyřešen v Unitboxu s využitím nové IoU loss funkce pro výpočet ohraničujícího rámce, které zpracovávají rámec jako celou jednotku. S použitím nové IoU loss a hloubkové CNN UnitBox dosáhl přesné a efektivní lokalizace, která byla dosažena na objektech s variabilním tvarem.

Mezi další konvoluční neuronové sítě patří:

- DenseBox;
- ExtremeNet;
- CornerNet;
- CenterNet;
- FoveaBox.

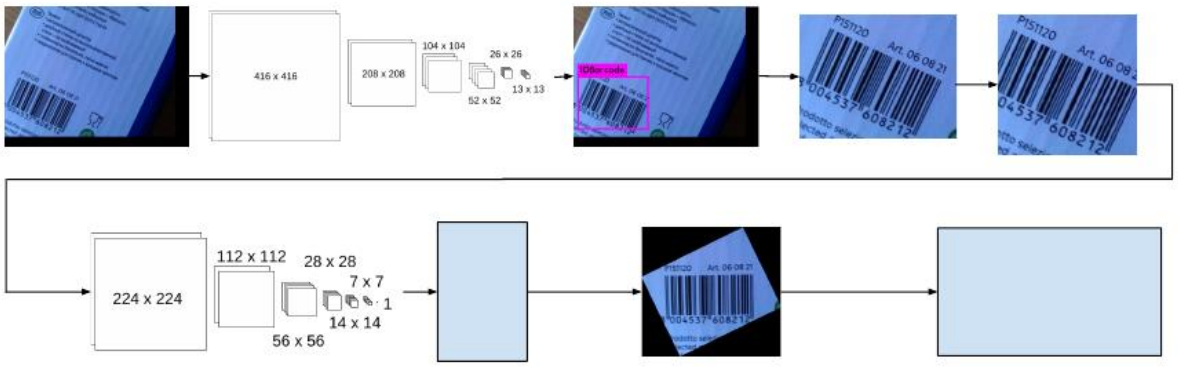
2.4 Neuronové sítě pro odhad úhlu a pózy

Většina neuronových sítí se zabývá jen lokalizací objektu v obrazu. S odhadem úhlu se dá setkat v topograficky orientovaných neuronových sítích, které zpracovávají satelitní snímky. Tyto sítě mají na výstupu pět hodnot, zatímco předchozí mají jen čtyři. Těchto pět hodnot představuje $\mathrm{xMin}, \mathrm{yMin}, \mathrm{xMax}, \mathrm{yMax}, \mathrm{angle}$.

Příkladem takové neuronové sítě je DRBox [5]. Je to síť zaměřená na snímky, na kterých jsou objekty orientovány do stran. Tato síť používá Caffe a SSD. Mezi další podobná řešení patří RSDet.

Obrázek 6: Ukázka DRBoxu [6]

Toto zpracování satelitních snímků představuje nejčastější uplatnění těchto sítí s detekci nalezené v prostředí internetu. Dalším uplatněním je například automatická korekce natočených obrázků nebo odhad natočení kol automobilu. Nejčastěji je tato neuronová síť rozdělená na dvě části. První část je klasická neuronová síť, která se stará o detekci objektů a odhad úhlu má za úkol druhá neuronová síť. Příkladem tohoto přístupu je DRBox, která je popsaná ve zdroji číslo 5. Označení jejich neuronové sítě jsem v textu nebylo napsáno. Jejich přístup používá YOLO algoritmus pro detekci QR kódu [4] a pak menší neuronovou síť, která daný QR kód natočí směrem k uživateli. Tato dvoufázová struktura byla použita jako inspirace při vytváření vlastní implementace neuronové sítě v TensorFlow 2.0 a při úpravě YOLO implementace.

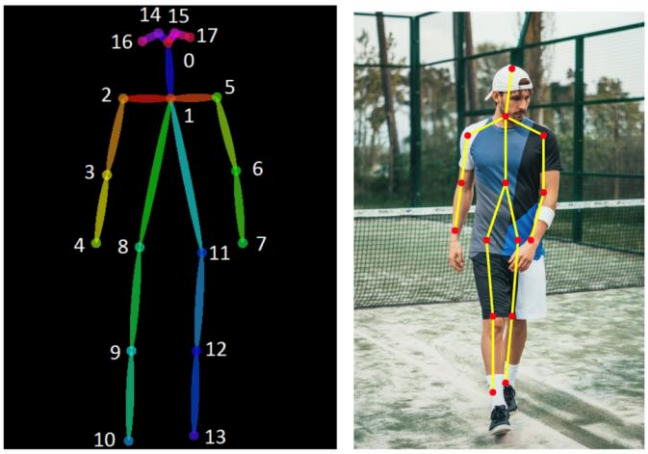


Obrázek 7: Princip neuronové sítě s detekcí úhlu [7]

Tyto neuronové sítě se nedaly plně použít pro řešení zadaného problému, protože odhadují úhel natočení 2D objektu ve 2D obrázku, ale zadaný problém je založený na odhadu 3D úhlu ve 2D obrázku. Ze začátku byly vyhledávány sítě na predikci úhlu, ale na konci ledna byly nalezeny sítě na predikci pózy. Nejčastější použití je na detekci lidí a jejich pózy v obraze. Detektor se snaží najít v obrázcích nebo videích spoje mezi částmi lidského těla, jako jsou například spojení ruky a trupu, trupu a nohy. Tyto sítě nejčastěji řeší pózu člověka, ale jsou i sítě, které jsou trénovány na detekci objektů. Díky faktu, že detekce pózy vyžaduje 3D objekty, se velice podobá problému zadanému v diplomové práci. Lze ho popsat jako problém nalezení kloubů. Nalezení pózy člověka má několik

podúloh jako odhad pózy s vyšším počtem lidí, odhad pózy ve skupině lidí, odhad pózy ve videu a odhad pózy jedné osoby. Mezi praktické možnosti uplatnění patří:

- rozpoznávaná aktivity,
- animace,
- počítačové hry,
- rozšířená a virtuální realita,
- robotika,
- bezpečnost.



Obrázek 8: Ukázka detektoru pózy člověka [8]

Příkladem může být aplikace HomeCourt, která používá detekci pózy hráče k analýze pohybu basketbalového hráče. Odhad pózy je velmi náročný, protože klouby jsou často špatně viditelné, nebo je špatné osvětlení. Oblečení snadné detekci také nepomáhá.

Tyto neuronové sítě lze také rozdělit podle principu práce. První skupina pracuje zdola-nahoru a druhá nahoru-dolů. V řešení zdola-nahoru je proces prováděn nejdříve ve velkém rozlišení a následně v malém. U druhého případu opačně. Princip zdola-nahoru začíná identifikací sémantických entit, které následně spojuje do instancí lidí. Druhý princip lokalizuje lidi a následně odhadne pózu. Jako další lze uvést rozdělení podle množství testovaných dimenzí na dvojdimenzionální a trojdimenzionální. Dvojdimenzionální hledá souřadnice (x, y) a trojdimenzionální hledá souřadnice (x, y, z).

Postup podobný nebo stejný jako detektor pózy člověka nelze aplikovat na objekty. Na detekci objektů existují speciální neuronové sítě, jejichž příklady jsou uvedeny v následující části:

DeepPose [12] je přístup od dvou autorů, kteří zároveň pracují u firmy Google. Samotný odhad pózy je založený na hluboké neuronové síti, která vyhledává klouby. Použití hluboké neuronové sítě bylo na danou dobu průlomové pro řešení těchto problémů. Jedná se o state-of-the-art řešení na standardních datasetech. DeepPose nepotřebuje grafické modely pro nalezení kloubů. Model se skládá z AlexNetu se speciální závěrečnou vrstvou, která generuje 2k koordinátů $(x_i, y_i) \times 2$ pro $i = \{1, 2, 3, \dots, k\}$. Model pro trénování používá L2 loss funkci. Použití kaskádového regresoru umožňuje dosáhnout velké přesnosti.

ConvNet je způsob, který v současné době využívá většina implementací řešení tohoto problému. Tato implementace se zabývá nalezením kloubů v monokulárním RGB obrázku. Model má oproti jiným větší pooling. Pro zobrazení výsledků hledání kloubů se používá tepelná mapa, která se generuje s pomocí algoritmu sliding window. Rozdíl oproti jiným implementacím je v tom, že se používají standardní konvoluční vlastnosti. Toto řešení redukuje počet trénovacích parametrů a reguluje model tepelné mapy. Pro shrnutí se model skládá z lokalizace založené na tepelné mapě, modulu pro zpracování konvolučních vlastností pro nalezení každého kloubu a konvolučním modelem pro úpravy.

OpenPose [18] je jedna z nejpopulárnějších knihoven, která používá přístup zdola-nahoru pro detekci a zjištění pózy velkého množství lidí v jednom obrázku. Jako ostatní implementace tohoto typu se v prvním kroku najdou klíčové body každé osoby na obrázku a následně se spojí podle jednotlivých postav. OpenPose neuronová síť v prvním kroku zjistí vlastnosti z obrázku s použitím prvních vrstev sítě. Tyto vlastnosti jsou poslány do dvou paralelních větví konvolučních vrstev. První větev odhadne mapu jistoty a druhá spoje mezi jednotlivými částmi. Výsledky obou větví se následně spojí dohromady.

Jako další sítě řešící tento problém bych uvedl:

- DeepCut;
- RMPE;
- Mask RCNN.

Výše uvedená řešení se používají pro detekci lidí, ale nehodí se pro problém detekce objektů, tím se zabývají jiné neuronové sítě, které jsou uvedy dále. Následující uvedené implementace se zaměřují na detekci 6D pózy objektu v prostoru. Uplatnění tyto implementace mají například v robotické výrobě, při které robotické rameno potřebuje znát přesné umístění a natočení objektu. V rozšířené a virtulní realitě pro snadnou interakci objektů s uživatelem. Příkladem je aplikace Ikea place od firmy IKEA, která umožní přes displej mobilního zařízení umístit nábytek do místnosti bez nutnosti zakoupení daného nábytku. Tyto implementace často používají RGB-D obrazy objektů bez textury nebo s texturou. RGB model doplněný o hloubku. Obrázky a videa s hloubkou se dají pořídit například z kamery, která je doplněná o hloubkový senzor. Mezi tyto kamery patří například Asus Xtion PRO LIVE. Rozšíření RGB modelu o hloubku umožnuje zvětšit množství příznaků a informací, které se dají zpracovat algoritmem a zároveň použít pro zobrazení výstupu. 2D výstup se posouvá do 3D a to umožnuje kolem 3 os otáčení a to dává 6 stupnů volnosti. Tyto technologie se začínají více používat ve videoherním průmyslu, ikdyž hardwarové nároky jsou stále vysoké. Mezi videohry které podporují rozšířenou a virtuální realitu lze zařadit Half-Life: Alyx, The Forest, War Thunder, PAYDAY 2, Pokemon GO a Harry Potter: Wizards Unite.

HybridPose [8] je v době psaní diplomové práce nejnovější state-of-the-art příspěvek do oblasti detekce 6D pózy objektu. Neuronová síť pro rozpoznání objektu je v šestidimenzionálním prostoru. Model vezme obrázek a předpoví pro něj vlastnosti, krajní vektory a rozdíl aktuální pozice proti standardní pozici. Použití okamžité reprezentace objektové pozice zlepšuje stabilitu modelu predikcí. Standardní postup pro šestidimenzionální rozpoznání využívá jedinou reprezentaci k uložení dat o póze, v tomto se HybridPose liší a používá okamžité zobrazení, které ukládá informace o geometrii objektu jako například hrany vektorů, klíčové body a odchylka od standardní pozice. HybridPose na Occlusion-LineMOD datasetu dostává přesnosti 79,2% a to představuje zlepšení o 67,4% oproti DPODu. HybridPose má velké zlepšení oproti implementacím, které za zaměřují jen na klíčové body.

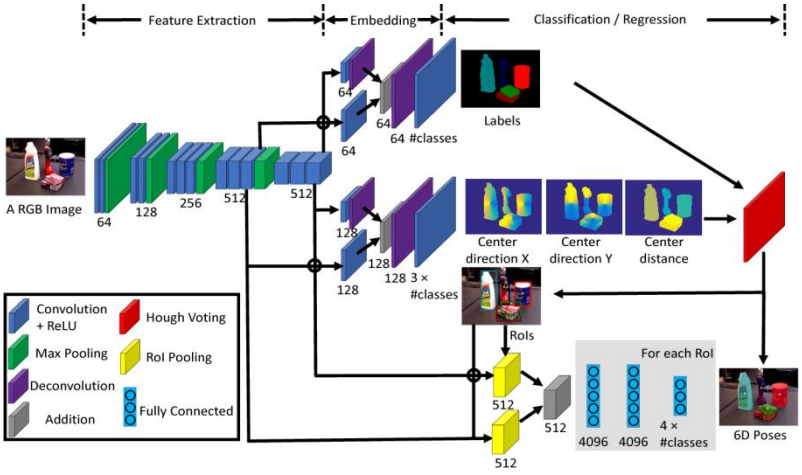
Vstupem sítě je obrázek z kamery a výstupem je šestidimenzionální umístění objektu relativně ke kameře. HybridPose používá 2 moduly prosvoji funkčnost. První predikční modul obsahuje 3 neuronové sítě na odhad klíčových bodů, množinu hran mezi klíčovými body a množinu korespondence podobností mezi pixely vstupního obrázku. Druhý modul se stará o regresi pózy. Optimalizuje pózu, aby byla kompatibilní s výstupy 3 neuronových sítí z prvního modulu. Modul kombinuje trénovatelné inicializované sub-moduly a trénovatelné zdokonalovací sub-moduly. Do prvního modulu se při trénování vloží trénovací data a do druhé validovací. Řešení má lepší výsledky než například konkurenční PoseCNN.

DPOD [16] je řešení inspirované Gaulerem a Taylorem, které odhaduje husté korespondence mezi modelem lidského těla a lidským tělem na vstupním obrázku. Na rozdíl od DensePose není potřeba pro DPOD vytvářet anotace ručně, ale anotace jsou vygenerovány počítačově do podoby ultrafialové texturové mapy. Mezi 2 hlavní body implementace DPOD patří pixelový odhad multitřídového objektu a klasifikace korespondenčních map, které udávají vztahy mezi pixely a 3D modelovými vektory. Implementace je trénována na kombinaci syntetických a reálných dat, protože tato kombinace dosahuje nejlepších výsledků.

SSD6D [17] je řešení jehož cílem je trénování na syntetickém datasetu, dekompozice pozového prostoru pro usnadnění trénování a rozšíření SSD pro detekci dvoudimenzionální detekce s šestidimenzionálním odhadem pózy. Program pro svou funkčnost využívá RGB-D data. Která podle tvrzení autorů zajištují skoro perfektní detekce.

PoseCNN [9] odhaduje trojdimenzionální transakci objektu přes lokalizaci prostřední části obrázku a odhadu jeho vzdálenosti od kamery. Trojdimenzionální rotace objektu je odhadnuta díky regresi na čtvercovou reprezentaci. PoseCNN využívá novou loss funkci, která umožňuje síti symetrické obrazy.

Hlavní princip používaný pro řešení je rozdělení problému do několika komponent, které umožňují síti modelovat závislosti a nezávislosti mezi nimi. Prvním krokem je odhad značky pro každý pixel ve vstupním obrázku. Druhým krokem je odhadnutí pixelových souřadnic objektu uprostřed přes odhad jednotkového vektoru od každého bodu ke středu. Síť také odhaduje vzdálenost středu objektu. Střed objektu a pozice kamery umožňuje vypočítat trojdimenzionální přechod. Posledním krokem je odhad trojdimenzionální rotace přes regresi konvolučních vlastností v prostředí ohraničujícího rámce na čtvrtinovou reprezentaci.



Obrázek 9: Architektura PoseCNN [9]

Síť obsahuje dvě části, první se skládá ze třinácti konvolučních vrstev a čtyř max-pooling vrstev, které extrahují mapu vlastností s různými rozlišeními vstupního obrázku. Tento krok tvoří základ dalších kroků. Dalším krokem je provedení tří úkolů, které vedou k vytvoření šestidimenzionálního odhadu pózy objektu.

RotationNet [11] má základní princip natrénování konvoluční neuronové sítě díky fixnímu počtu obrázků, které reprezentují rotaci objektu ve specifickém pořadí. Během trénování se póza pokládá za latentní vlastnost, to znamená, že je odhadnuta díky neřízenému trénování. Po trénování se obrázky jednoho objektu pošlou do sítě, aby došlo k predikci pózy a třídy objektu. Trénování probíhá s předtrénováním na Imagenet datasetu, ale výstup je upraven a zpracován na pořadí pohledů.

Tyto neuronové sítě byly nalezeny na začátku února, když už byl odevzdaný přístup z většiny dokončený. Proběhlo vyzkoušel implementace RotationNet v knihovně PyTorch, kterou se nepodařilo spustit kvůli nízké hodnotě Compute Capability grafické karty v pracovním notebooku.

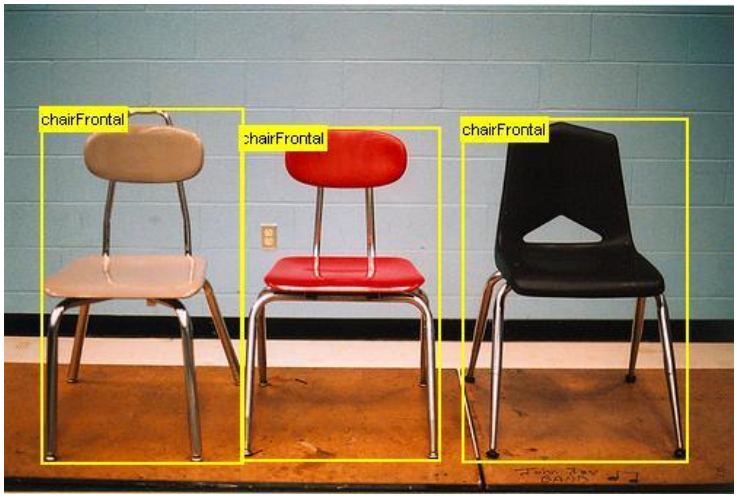
2.5 Datasety

Datasety jsou klíčovou částí strojového učení. Velký vývoj v té této oblasti má dopad na výsledky samotných algoritmů. Pro efektivní trénování je potřeba mít správně označený dataset trénovacích a testovacích dat. Tyto datasety je obvykle těžké získat z prostředí reálného světa, proto se vývoj zaměřil na generování datasetů synteticky. Datasety bychom mohli rozdělit do skupin na:

- obrázková data: pro rozpoznávání obličeje a akce, detekci objektů v obraze, rozpoznání objektů z leteckých snímků, rozpoznání ručně psaného textu a rozpoznání písmen;
- textová data: recenze, články, zprávy, chatové konverzace;
- zvuková data: mluva, hudba, zpěv, zvuky zvířat;
- signálová data: elektrické signály, záznam pohybu;
- fyzická data: astronomie, zemská fyzika, systémy, vysokoúrovňová fyzika;
- biologická data: lidí, zvířat, rostlin, virů, bakterií, mikroorganismů.

Pro neuronové sítě, které se učí vyhledávat obecné a časté prvky reálného světa, existuje množství datasetů, které se dají použít pro tuto problematiku. Pro kontrast průmyslový problém detekce inspekce kvality produktu jsou těžko k nalezení. Trénování neuronových sítí znamená najít správné hodnoty pro její volné parametry s použitím řízeného učení. Tato operace potřebuje velké množství dat pro trénování. Pro potřeby diplomové práce se zabývá obrázkovými daty a detekcí objektů, tato oblast má několik vytvořených datasetů.

- Pascal VOC: Cílem tohoto datasetu je rozpoznávat třídy objektů z velkého množství obrázků reálných scén. Obsahuje 20 tříd objektů. Mezi ně patří: osoba, pták, kočka, pes, kůň, ovce, letadlo, kolo, loď, autobus, auto, motorka, vlak, láhev, židle, stůl, gauč, televize a květina v květináči. dataset se zaměřuje na lidi, osoby, dopravní prostředky a obvyklé vybavení domácnosti. Poslední verze z roku 2012 obsahuje 11 530 obrázků pro trénování a validaci. V obrázcích je 27 450 anotovaných objektů a 6 929 segmentací. Dataset se zaměřuje na problém klasifikace, detekce a segmentace. Další vývoj datasetu se neočekává, protože hlavní osoba, která se podílela na vývoji (Mark Everingham), zemřela.



Obrázek 10: Ukázka z Pascal VOC [10]

- COCO: Je obsahově velký dataset který se zabývá problémem detekce, segmentace a klasifikace objektu. Dataset obsahuje 80 tříd, mezi ně patří: lidé, batoh, deštník, kolo, autobus, auto, kočka, pes, slon, drak, pálka, sklenice, lžička, ovoce, televize, notebook a lednička. Podle kategorií se třídy dělí na – člověk a cestovní věci, dopravní prostředky, věci na ulici, zvířata, sportovní pomůcky, jídla a pití, ovoce a vybavení domácnosti. Poslední verze vyšla v roce 2019. COCO obsahuje 330 000 obrázků a z nich je 200 000 označených. Dále dataset obsahuje 91 kategorií věcí a 1 a půl milionu objektových instancí.

Obrázek 11: Ukázka z COCO [11]

- Image Net: Jedná se o velmi obsáhlou databázi obrázků organizovaných podle WordNet hierarchie. Každý koncept je rozdělen podle slov nebo slovních frází do synsetů. Celkem je v datasetu více než 80 000 synsetů. V současnosti mají přes 500 obrázků pro každou třídu, ale dataset ještě není kompletně dokončen. V datasetu je celkem 200 tříd. Ve verzi z podzimu roku 2011 se nachází celkem 32 326 obrázků rozdělených do kategorií: květiny, flora, geologické formace, přírodní objekty, sport, artefakty, fungus, osoby, zvířata a ostatní.
- ObjectNet: Jedná se o unikátní dataset kvůli detailu, že neobsahuje trénovací, ale jen testovací obrázky. Obrázky byly pořízeny s cílem zachytit objekty z nových pohledových úhlů a na nových pozadích. Celkem obsahuje 50 000 obrázků s měnícím se pozadím, orientací a úhlem pohledu. Počet tříd v datasetu je 313, z toho 113 je stejných jako v ImageNetu.

- Open Images: Jedná se o obsáhlý dataset, který tvoří velké množství obrázků anotovaných s označením třídy, ohraničujícími rámci a segmentační maskou. Obsahuje celkem 16 milionů ohraničujících rámců pro 600 tříd a 1,9 milionů obrázků, což tento datový rámec dělá největším existujícím datasetem s objektovou lokalizační anotací.

všechny následující datasety jsou z [19].

- MIRO je dataset pro neuronové sítě s 6D detekcí pózy objektů, který je zaměřený na objekty každodenního používání. Jedná se o dataset který je určený pro RotationNet. Obsahuje obrázky které se dělí do následujících skupin: autobus, auto, čistící prostředek, hodiny, hrnek, sluchátka, myš, nůžky, bota, sluneční brýle, koník a řezač lepící pásky. Každá třída obsahuje přes 100 obrázků daného objektu z různých pozic. Obrázek nemá připojený žádný označovací soubor, protože označení jsou uloženy v názvu obrázku.
- T-LESS je dataset pro neuronové sítě s 6D detekcí pózy objektů, který je zaměřený na objekty používané v průmyslové oblasti. Obsahuje 30 různých objektů bez textury. Každý objekt je definován 1296 templaty a 16 bitovými hloubokými mapami. Templaty zachycují objekt z různých úhlů na horní hemisféře s 10 stupňovým krokem elevace a 5 stupňovým azimutem.
- LineMOD je další dataset pro neuronové sítě s 6D detekcí pózy objektů, který je zaměřený na objekty každodenního používání. Obbjekty jsou uloženy bez textury. V datasetu se nachází přes 18 000 obrázků rozdělených do 15 typů.
- HomebrewedDB je nejnovější dataset pro 6D odhad pózy objektu. Obsahuje 33 typů objektů. Jedná se o hračky, věci pro domácnost a průmysl. Objekty jsou uloženy bez texury jako u minulého datasetu.

2.6 Datasety pro trénování a testování

K zadání diplomové práce byl dodán referenční dataset, který se skládá ze syntetických obrázků Lego stavebnice vesmírné lodi Alienator. Jedná se o 82dílkovou stavebnici, která byla vydána v roce 1987. Každý obrázek obsahuje Alienátora uprostřed obrázku a pozadí, které je proměnné a vyfocené v prostředí reálného světa. Alienátor je zobrazen z několika vertikálních (úhlů sklonu) a horizontálních (úhlů natočení) úhlů. Díky tomuto datasetu je možné natrénovat neuronovou síť, která bude umět rozpoznat Alienátora z každé strany kromě spodní. Každý obrázek byl vygenerován synteticky a uložen jako .png v rozměrech 640 × 480 pixelů a bitovou hloubkou 32. 640 pixelů na šířku a 480 pixelů na výšku. Dataset doplňují obrázky pozadí a jiných objektů, které jsou označené jako chybné. Cílem je naučit síť rozpoznávat chybné obrázky a prázdné obrázky a označit je jako chybné.

Obrázek 12: Ukázka obrázku z trénovacího datasetu

Ke každému .png obrázku je přidán .exr soubor. Jedná se o HDR soubor, který obsahuje informace o pozadí a popředí v přiloženém .png obrázku. Tento .exr soubor je velmi užitečný pro označovací software, který s jeho pomocí udělá jednoduše ohraničující rámec, který se použije pro vytvoření anotovacích .txt a .xml souborů. Dalším uplatněním je snadná výměna pozadí díky informacím z tohoto souboru.

Obrázek 13: Ukázka obrázku z testovacího datasetu

Testovací dataset obsahuje reálně pořízené obrázky Alienátora nejčastěji z přední a levé strany. Testovací obrázky jsou uloženy v rozměrech 640 × 480 pixelů a bitovou hloubkou 24, nebo 96. 640 pixelů na šířku a 480 pixelů na výšku. K testovacím obrázkům nebyl přiložen .exr soubor, s jehož pomocí se automaticky generují označení v pomocném programu. Vytvoření označovacích souborů bylo provedeno ručně použitím programu YOLO BBox Annotation Tool.

Tesně před odevzdáním proběhla změna testovacího datasetu, kdy byla část testovacích obrázků smazána a nahrazena novými, které se dají lépe použít pro generování výsledků. Ve staré verzi byla velké množství velmi si podobných obrázků s Alienátore, které se lišili natočením Alienátora o jeden stupeň natočení. Tyto obrázky byly promazány a množství obrázků zůstal jeden. V kapitole výsledků jsou uvedeny výsledky pro novou i starou verzi testovacího datasetu.

2.7 Syntetické a reálné obrazy

Důležitým bodem diplomové práce jsou syntetické a reálné obrazy, mezi nimiž je podstatný rozdíl. Reálné obrazy jsou generovány ze zdroje reálného světa. Vznikají na principu koncentrace světla, nebo elektromagnetických vln v rozmezí mezi ultrafialovým a infračerveným na dvojdimenzionální plochu přes optický systém. Mezi reálné obrazy můžeme zařadit satelitní snímky, vzdušné obrazy, zdravotnické obrazy, mikroskopické obrazy a fotografie. Do této kategorie se dají přidat také videa, ze kterých se následně dají vygenerovat snímky. Pořídit reálný obraz lze například pomocí fotoaparátu nebo videokamery.

Syntetické obrazy se dají popsat jako dvojdimenzionální pole dat, kde hodnoty pro každý element neboli pixel se převádějí na intenzitu nebo barvu, která se následně zobrazí. Syntetická i reálná data se ukládají v počítačových a mobilních zařízeních jako dvojdimenzionální pole hodnot a zobrazují se jako obrázky, ale rozdíl je v tom, že hodnoty v poli reálných obrazů korespondují se signálem produkovaným senzory, jež byly ozářeny elektromagnetickou vlnou v optickém systému. Pro kontrast hodnoty v poli syntetických obrazů jsou vypočítávány na základě jiného typu signálu. Příkladem optického obrazu je CAT sken, radarové obrazy, elektronové mikroskopové obrazy a obrazy generované pomocí programovaného generátoru paprsků ve virtuálním prostředí. Cílem syntetického renderování je mít co nejlepší kvalitu syntetického obrazu, aby byl nerozpoznatelný od reálného obrazu vyfoceného ve stejné scéně, a použít co nejvíce přírodních zákonů jako lom světla a odraz světla pro dosažení co nejlepšího obrazu, který se co nejvíce blíží fotografii pořízené v reálném prostředí, reálné scéně.

2.8 Syntetické obrazy pro neuronové sítě

Potřeba velkého počtu anotovaných datasetů a obrázků různých tříd pro trénování konvolučních neuronových sítí (CNN) byla vždy překážkou pro jejich plné využití v aplikacích počítačového vidění. Trénování neuronových sítí bylo často založené na datasetech vzniklých z reálných obrazů, jejichž pořízení a shromáždění je časově náročné, proto se zkoumala možnost trénování neuronových sítí datasetem vzniklým synteticky, druhým testovaným přístupem bylo rozšíření datasetu díky transformaci, která zachovala označení. Vytvoření těchto datasetů je jednodušší, protože není potřeba snímky sbírat v prostředí reálného světa. Anotace reálných dat je rovněž časově velmi náročný úkol. Datasety mohou mít desetitisíce obrázků s velkým množstvím různých tříd. Ruční anotace datasetu diplomové práce zabrala hodinu. Trénování neuronových sítí syntetickými obrazy je nový přístup pro detekci objektů v oblastech, kde je těžké sehnat dostatečné množství reálných dat, například v kategorii detektoru velkého množství proložených objektů.

Hlavními doménami, kde nalézá uplatnění využití trénování neuronových sítí s pomocí syntetických obrazů je:

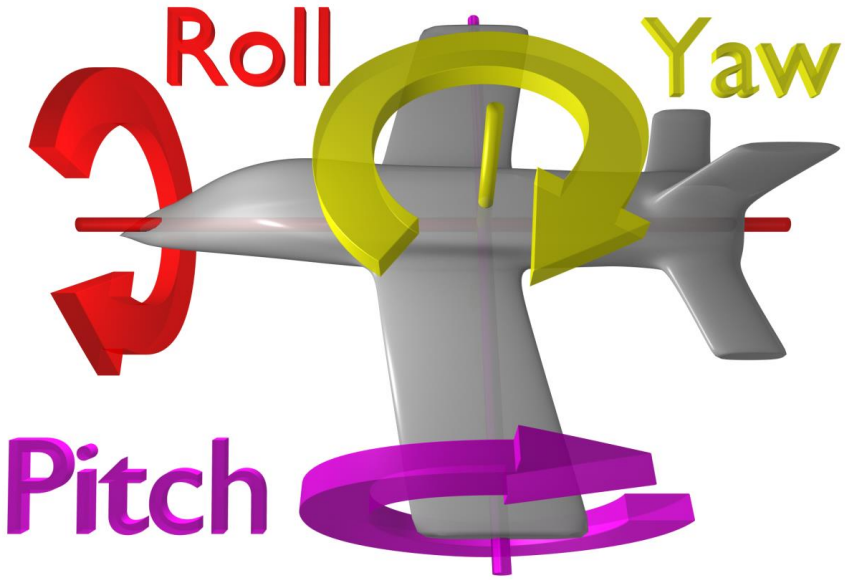
- Nalezení pozorovacího bodu,
- Odhad pózy,
- Sémantická segmentace,
- Detekce objektu,
- Rozdílnost a odhad optického toku.

2.9 Popis úhlů

V této podkapitole je vedena informace o použitých úhlech. Celkem lze použít pro rotaci 3 typy úhlů:

1. úhel natočení (yaw)
2. úhel sklonu (pitch)
3. úhel naklonění (roll)

V anotovacích souborech jsou uloženy dva typy úhlů, z nichž objekt vidíme: horizontální (úhel natočení) a vertikální (úhel sklonu) úhel natočení objektu. Oba úly je možné použít pro natrénování detektoru. Informace který úhel se použije je uložena v konfiguračním souboru. Pro generování výsledků byl použit úhel natočení.



Obrázek 14: Typy úhlů [14]

3. Použité knihovny

Před popisem samotné aplikace je dobré přiblížit v této kapitole důležité knihovny, které byly použity pro vytvoření diplomové práce, a jak vypadají označovací soubory pro datasety a možnosti, jak tyto soubory vygenerovat.

3.1 Open CV

Open CV je volně dostupná knihovna pro práci s obrazem v jazycích C/C++, Python a Java. Podporuje zařízení s operačním systémem Windows, Linux, Mac, iOS a Android. Open CV byla vytvořena se zaměřením na real-time aplikace. Vydávána je pod licencí BSD. Jedná se o knihovnu zaměřenou na manipulaci s obrazem, počítačové vidění a zpracování obrazu v reálném čase. Původně byla vyvíjená americkou firmou Intel. OpenCV podporuje knihovny pro hluboké učení jako TensorFlow, PyTorch a Caffe.

3.2 Tensorflow

Tensorflow je volně dostupná open-source knihovna pro práci s datovým tokem a diferencionálním programováním. Jedná se o symbolickou matematickou knihovnu a také knihovnu pro strojové učení jako například neuronové sítě. Tato knihovna byla vytvořena týmem z americké firmy Google a byla zveřejněna v roce 2015, lze ji použít v programovacích jazycích Python a C++. Knihovna podporuje grafické výpočty přes technologii Nvidia Cuda. Podporované operační systémy jsou Linux, Windows, Mac OS a Android. Jméno vzniklo z operací, které neuronové sítě provádějí nad vícerozměrnými datovými poli, kterým se říká tensory. Mezi příklady použití TensorFlow je automatický software pro ukládání obrázků jako například DeepDream. RankBrain představuje nový vyhledávací nástroj, který umí zpracovat větší množství vyhledávání. Cílovou skupinou pro tuto knihovnu jsou programátoři a vědci.

TensorFlow mimo knihovny nabízí také komplexní a flexibilní ekosystém nástrojů, knihoven a komunitních zdrojů. Mezi uživatele této knihovny patří Airbnb, Coca Cola, Twitter, Intel a DeepMind.

V lednu 2019 byla představena vylepšená verze TensorFlow 2.0, která byla vydána v září toho samého roku a přinesla velmi významné změny. Velkým problémem hlavně pro začínající uživatele knihovny bylo její velmi komplikované API, jež vedlo mnoho lidí k použití jednodušších API, které používaly TensorFlow jako Keras a TF Slim. To bylo ve verzi 2.0 změněno. Došlo k vyčištění a úpravě částí API. Byl zrušen tf.app, tf.flags a nahrazen částí z tf.contrib. Některé málo používané funkce byly přesunuty do tf.math. a některé části byly nahrazeny za jejich 2.0 ekvivalent, například tf.summary, tf.keras.metrics. Keras se stal součástí samotného TensorFlow. Pro konverzi kódu ze staré verze na novou je k dispozici skript přímo od TensorFlow týmu.

Další důležitou a vítanou změnou je použití dychtivého spuštění, které používá Python. Stará verze požadovala po uživateli, aby manuálně kompiloval syntaktický strom a zavolal session.run. V nové verzi se sessions volají automaticky.

Starší verze velmi používala globální jmenné prostory. Tf.Variable jsou umístěny do grafu a tam jsou uloženy, přestože Python k nim ztratil odkaz. Načtení této hodnoty je možné, jen pokud programátor zná jméno, pod nímž byla vytvořena. Ve staré verzi byly implementovány metody, které pomáhají

uživateli s tímto problémem jako tf.get_global_step(). Nová verze plně eliminuje tyto mechanismy. Ztracené tf.Variable jsou automaticky chyceny Garbage Collectorem a odstraněny.

Ve verzi 2.0 je možnost obarvit funkce použitím tf.function() k označení pro JIT (Just-in-time) kompilaci, aby TensorFlow běželo jako jeden graf. Toto umožňuje získat lepší optimalizaci a přenositelnost. Uživatelé díky tomu mohou zkombinovat expresivitu Pythonu a přenositelnost Tensorflow mezi Pythonem, C++ a JavaScriptem. Pro automatickou konverzi lze použít AutoGraph.

3.3 Anotace datasetů

Anotace datasetů je důležitý krok v přípravě dat. Anotovací soubor obsahuje informace o třídě objektů, které se nalézají v přiloženém soubory a jejich koordinační souřadnice. YOLO používá jednoduché schéma uložené v .txt souboru:

```
<object-class> <x_center> <y_center> <width> <height>
```

- object class: celé číslo reprezentující třídu v rozmezí 0-N kde N je počet tříd-1
- x_center, y_center, width, height: float čísla relativní k výšce a šířce obrázku v rozmezí 0,0–1,0
- příklad: height = height_of_bounding_box / image_height

```
1 0.716797 0.395833 0.216406 0.147222
```

Coco dataset používá pro anotaci .json soubor. Anotace je ve tvaru:

`"info": info,` obsahuje vysokoúrovňové informace o datasetu

`"licenses": [license],` obsahuje list licencí obrázků

`"categories": [category],` obsahuje list všech kategorií

`"images": [image],` obsahuje list všech obrázků bez ohraničujících rámců a segmentačních informací. ID obrázku musí být unikátní

`"annotations": [annotation]` obsahuje list všech anotací pro každý obrázek v datasetu, tato část obsahuje ohraničující rámce a údaje o segmentaci

Ohraničující rámec je v COCO datasetu v následujícím formátu:

```
(x_top_left, y_top_left, width, height)
```

Pascal VOC dataset používá pro anotaci .xml soubor. Anotavaný .xml soubor použitý v diplomové práci je založen na Pascal VOC schématu, jako příklad je zde uvedena anotace použitá v diplomové práci:

```
<annotation>
<folder>alienator</folder>
<filename>6876_alienator_002.png</filename><path/>
<source><database>unknown</database></source>
<size><width>640</width>
<height>480</height>
<depth>3</depth></size>
```

```
<segmented>0</segmented>
<object><name>left Alienator</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<horizontal>0.750000</horizontal>
<vertical>0.250000</vertical>
<bnbbox><xmin>129</xmin><ymin>131</ymin><xmax>555</xmax><ymax>344</ymax>
</bnbbox></object></annotation>
```

Ohraničující rámec je v Pascal VOC datasetu ve formátu:

```
(x_top_left, y_top_left, x_bottom_right, y_bottom_right)
```

Rozdíl mezi anotacemi v těchto datasetech je, že v Pascal VOCu je vytvořen anotovaný soubor pro každý obrázek v datasetu. V COCO datasetu je vše umístěno v jediném souboru. Ohraničující rámce a anotovací soubory se liší.

3.3.1 Anotovací softwarové nástroje

Anotace datasetů lze rozdělit do dvou skupin:

- ruční anotace,
- algoritmická anotace.

Ruční anotace je časově velmi náročný proces, který se významně prodlužuje s velikostí datasetu. Autor použije program, který mu umožní v obrázku vytvořit ohraničující rámec a ten se následně uloží do označovacího souboru ve vybraném formátu. V prostředí internetu se nachází množství programů, které umožňují ruční označení obrázků a následné vygenerování označení v různých formátech. Výhodou tohoto přístupu je velká možnost kontroly, že se objekt označil ve správné velikosti a se správnou třídou. Mezi výhodami se dá také zmínit, že po označení všech obrázků lze vygenerovat různé typy označovacích souborů za sebou. Tento přístup je efektivní pro malé datasety.

Algoritmická anotace využívá program, který načte, zpracuje obrázky a vygenerované anotace uloží do souboru. Pro tento přístup je dobré mít předpřipravené pozice a třídy pro každý objekt, nebo použít algoritmy jako Selectice Search pro nalezení daných pozic objektů v obraze. K diplomové práci byly přidány obrázky i soubory, ze kterých se souřadnice bodů daly jednoduše zjistit. Výhodou tohoto přístupu je vysoká rychlost anotace po sepsání programovacího kódu. Na internetu je k dispozici velké množství knihoven pro práci s obrazem. Nejznámějším příkladem je OpenCV knihovna. Tento přístup je efektivní pro velké datasety.

Mezi ruční anotavací softwarové aplikace patří:

- [3] LabelImg je grafický označovací nástroj, který je napsaný v Pythonu a používá knihovnu Qt pro grafické rozhraní. Anotace se ukládají v Pascal VOC formátu, nebo YOLO formátu. Pro použití této aplikace je potřeba mít v počítači nainstalovanou knihovnu Qt a programovací jazyk Python. Program funguje na unixových systémech, MacOS a Windows. Program je k dispozici zdarma na GitHubu.
- [4] YOLO BBox Annotation Tool je program vytvořený v Javascriptu, který používá rozhraní internetového prohlížeče, díky tomu je spuštění jednoduché a není potřeba cokoli instalovat. Tento program byl vytřen s použitím HTML, CSS a Javascriptu. Uživatelské rozhraní není stejně pěkné jako v případě LabelImg, ale funkcionalita je výborná. Program umožňuje vytvářet označovací soubory ve formátu COCO, YOLO a Pascal VOC. Program je zdarma ke stažení na GitHubu. I přes použití internetového prohlížeče je celá práce dělána lokálně, není potřeba žádná komunikace se serverem. Mezi vhodné vedlejší funkce patří tlačítko, které vystřižení a uložení ohraničujících rámců. Tento program byl použit pro ruční označení v diplomové práci.
- [5] VoTT od společnosti Microsoft je označovací program pro obrázky a videa. Jedná se o React a Redux Web aplikaci napsanou v jazyce TypeScript. Umožňuje načítání dat z lokálního i ze vzdáleného úložiště. Program lze nainstalovat jako samostatnou aplikaci, nebo ji spustit v prostředí webového prohlížeče. VoTT lze použít v operačním systému Windows, Linux a OSX. Tento program byl pro zpracování diplomové práce vyzkoušen, ale nevyhovoval. Program je zdarma ke stažení na GitHubu.
- [6] VGG Image Annotation Tool je jednoduchý označovací program, může být použit pro anotaci obrázků, videí a zvukových souborů. Tento program byl vytvořen s použitím HTML, CSS a Javascriptu a může být spuštěn v prostředí webového prohlížeče a umožňuje ukládání označovacích souborů ve formátu JSON, CSV a COCO. Tyto anotovací soubory nebyly pro diplomovou práci potřebné.

Mezi další programy lze zařadit Labelbox[7], Supervise.ly [8] a LabelMe [9].

4. Rozbor přístupu a popis aplikace

Po představení použitých informačních technologií, syntetických obrazů a neuronových sítí se můžeme zaměřit na popis praktické části diplomové práce. První části se bude zabývat vytvářením označovacích souborů, poté příjde na řadu YOLO framework Darknet a na závěr neuronové sítě v Tensorflow 2. Zde budou popsány navržené neuronové sítě použité v YOLO implementaci s potřebnými úpravami, které v ní bylo potřeba udělat. Tyto úpravy budou také popsány.

Přes letní prázdniny roku 2019 probíhala příprava označovacích obrázků a výběr neuronových sítí. Podle doporučené literatury jsem se zabýval neuronovou sítí UnitBox, která vznikla ze sítě DenseBox a na niž potom navázal EAST. EAST je textový detektor, který hledá text v obrázcích. Z toho důvodu nebyla toto řešení ideální pro tuto diplomovou práci. Protože práce předpokládá využití existujícího frameworku, hledal jsem existující implementaci tohoto algoritmu. Našel jsem na GitHubu implementaci, ale jelikož byla ve starých verzích Pythonu, Tensorflow a dalších knihoven, nepodařilo se mi ji zprovoznit. Mé další kroky směřovaly k Darknet YOLu z doporučené literatury a s tímto frameworkem jsem pracoval do prosince, kdy jsem zjistil, že do frameworku se mi nepodaří implementovat rozpoznávání úhlu podle zadání. Rozhodl jsem se použít Tensorflow v nové verzi 2.0 a napsat vlastní neuronovou síť a pro porovnání upravit existující YOLO implementaci v Tensorflow ve verzi 2.0.

Cílem bylo:

- Integrovat funkčnost požadovanou na základě zadání diplomové práce.
- Porovnat alespoň 2 metody, které splňují zadání.

Vývoj probíhal postupně podle zobecněných kroků:

- Vyzkoušení UnitBoxu a příprava označení trénovacích dat.
- Vyzkoušení Darknet YOLO a zkoušení přidání úhlu do frameworku.
- Seznámení s Tensorflow 2.0 a vytvoření vlastní neuronové sítě inspirovanou již vytvořenou implementací.

Program pro automatické označení je vytvořený ve vývojovém prostředí Codeblock v jazyce C++. Darknetová implementace YOLa je v jazyce C. Závěrečné neuronové sítě jsou napsány v jazyce Python a spouštěny přes systémovou konzoli.

4.1 Označovací software

Označovací program, který je pojmenovaný jako labeling, je vytvořený jako konzolová aplikace v prostředí CodeBlocks IDE. Tato aplikace byla myšlena jako doplněk. Hlavní cílem bylo mít program, který rychle vygeneruje potřebné soubory pro trénování a upravené obrázky pro dataset. Mezi úpravy obrázku patřilo například slabé rozmazání a výměna pozadí okolo Alienátora. Pro porovnání správnosti vygenerovaných označovacích souborů byly označovací soubory vytvářeny paralelně také ručně v označovacím programu YOLO BBox Annotation Tool. Z hlediska rychlosti je automatické generování lepší. Z hlediska kontroly nad samotnými označeními je lepší ruční verze. Dataset je popsaný na předcházejících stranách.

Program se skládá z jednoho souboru, kde je uložený celý proces. Začátek souboru obsahuje klasický import všech potřebných knihoven a vytvoření jmenných prostorů. K chodu tohoto programu je potřeba importovat jedině OpenCV knihovnu.

Program začíná vytvořením pomocných proměnných pro přístup k souborům. Dále se vytvoří počítací proměná, která bude ukládat informaci o počtu zpracovaných obrázků, a v dalším kroku se vytvoří proměná pro práci s přiloženým .csv souborem a proměnná pro matice.

Hlavní částí programu je While cyklus, který se stará o načtení .exr obrázku a dat z CSV, které se následně použijí pro vygenerování označovacích souborů. Hned po začátku cyklu je načtení dat z přiloženého .csv souboru, ve kterém je pro každý obrázek informace o třídě objektu a horizontálním (úhlu natočení) a vertikálním (úhlu sklonu) úhlu. V cyklu dojde zároveň k úpravě obrázku podle nastavení. Následně z počítací proměnné vytvořím tříměstné číslo a to použiju pro vytvoření jmen souborů, které budu načítat, nebo ukládat. V této části se také načte obrázek který se následně upraví rozmazáním nebo výměnou pozadí z jiného vstupního obrázku pozadí. upravený obrázek se následně uloží aby odpovídal vygenerovaným označovacím souborům.

Po této přípravě se načte .exr soubor a provede se úprava obrázku na černobílou barvu a zavolá funkce OpenCV findContour, která v .exr najde popředí objektu. V .exr je vždy jeden objekt. Z tohoto objektu se s pomocí contours získají souřadnice pro ohraničující rámec. V dalším kroku dojde k vytvoření prvního anotovacího souboru .txt pro YOLO přesně ve formátu popsaném v sekci o označování.

Dalším krokem je vytvoření .xml souboru podle PASCAL VOC formátu, popsaném v sekci o označování. Tyto kroky ve While cyklu se provedou pro 222 trénovacích obrázků a algoritmus následně skončí.

4.2 Darknet

Název tohoto frameworku velice připomíná temnou oblast ve webovém světě, ale tento framework se zabývá konvoluční neuronovou sítí YOLO ve verzi 2 a 3. První verze byla vytvořena autorem pjreddie [1] a následně upravena autorem AlexeyAB [2]. Verze od Alexeye byla ta, která byla použita pro pokus o vyřešení zadání diplomové práce. Jedná se o framework napsaný v jazyce C, jenž lze spustit na zařízeních s operačním systémem Windows a Linux. Darknet je k dispozici zdarma na webu GitHubu. Testovaná verze byla YOLOv3. Na podzim minulého roku se jednalo o nejaktuálnější verzi této konvoluční neuronové sítě. 23. 4. 2020 byla zveřejněna novější verze YOLOv4 a tuto Darnetovou implementaci jsem už v rámci diplomové práce netestoval.

Pro spuštění je potřeba splnit požadavky:

- operační systém Windows nebo Linux,
- CMake >= 3.8,
- CUDA 10.0,
- OpenCV >=2.4,
- cuDNN >= 7.0 pro CUDU 10.0,
- GPU s Compute Capability >= 3.0.

V Darknetu lze použít velké množství datasetů jako COCO, OpenImages, Pascal VOC a další různé specializované datasety například pro dopravní značky a další. Velkou výhodou tohoto frameworku je možnost trénování na vlastním datasetu. Tuto možnost jsem využil v první části diplomové práce.

Framework se ovládá příkazy přes příkazovou řádku, grafické rozhraní nemá. Toto ovládání je typické v této kategorii aplikací. Grafické rozhraní je výjimka.

Instalace na Windows byla jednoduchá. Bylo potřeba nainstalovat Visual Studio, CUDA, cuDNN a OpenCV. Po instalaci potřebných součástí stačilo otevřít darknet.sln, nastavit verzi x64 a spustit Release.

Pro trénování nad vlastním datasetem bylo potřeba v prvním kroku vytvořit vlastní konfigurační soubor .cfg a správně nastavit údaje pro framework. Mezi důležité údaje patří:

- batch a subdivision size,
- max_batches,
- steps a velikost neuronové sítě,
- počet tříd v každé YOLO vrstvě a počet filtrů ve vrstvě před YOLO vrstvou. Vzorec pro počet filtrů je: $(\text{classes} + 5) \times 3$.

Po vytvoření tohoto souboru je potřeba vytvořit také .names a .data soubory. Names soubor obsahuje jména všech tříd a .data soubor obsahuje počet tříd, cestu k trénovacím a testovacím datům, odkaz na .names soubor a odkazy na backup. Obrázky s označeními je vhodné mít umístěné v záložce data. Data musí být označená v YOLO formátu.

Před trénováním je vhodné stáhnout předtrénované váhové soubory. Není to potřebný krok, ale použití předpřipraveného souboru zrychlí proces trénování a zajistí větší přesnost než v případě trénování bez něj. Tyto postupy jsem dodržel a natrénoval Darknet na vlastním datasetu. Trénování YOLO-tiny zabralo na pracovním notebooku 6 hodin. Výsledkům se budu věnovat v další kapitole.

Framework se skládá z velkého množství .h, .c a .cu souborů. Celkem se tam nachází přibližně 135 souborů. Tato značná velikost a vzájemná propojenost ztěžovaly orientaci ve frameworku a přípravu na přidání úhlu do neuronové sítě v něm. Po konzultaci jsem od tohoto směru upustil a začal se věnovat TensorFlow 2.0 implementaci, která nabízí velké možnosti úprav a práce s touto knihovnou bude jednodušeji pochopitelná.

4.3 Implementace v Tensorflow 2.0

V této části budou popsány TensorFlow soubory s neuronovými sítěmi, které jsou součástí odevzdané implementace. První soubor se bude zabývat detekcí a trénováním celým procesem trénováním detektoru pro lokalizaci a detekci úhlu. V části detekce úhlů se trénují 4 strany, které představují levou stranu, pravou stranu, přední stranu a zadní stranu. Druhý soubor bude využívat pro běh natrénované váhové soubory. Třetí soubor je soubor který obsahuje funkce a proměnné které jsou společné pro oba předchozí soubory. Tento soubor není určen ke spouštění, ale slouží pro zfeketivnění a zpřehlednění kódu. Který úhel se použije je uloženo v konfiguračním souboru ve složce configs. Jedná se o malé soubory pro každou ze 4 implementací, které obsahují 4 proměnné:

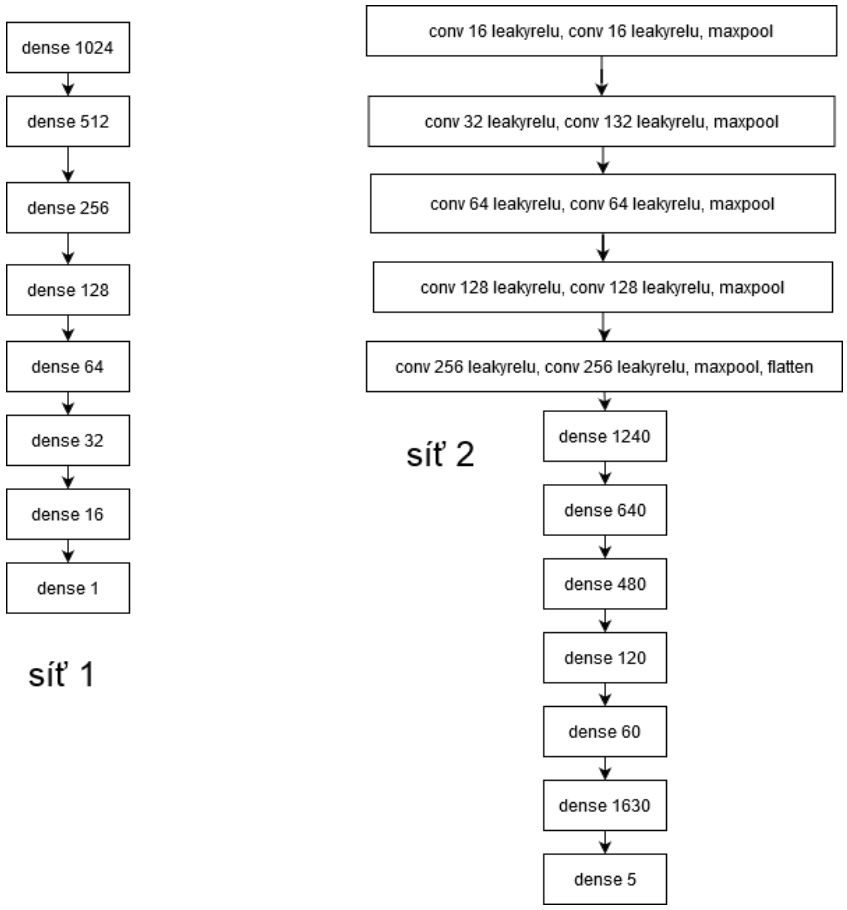
1. měřítko pro velikost obrázků pro síť která detekuje úhel natočení. Měřítko je určeno pro snadnou změnu velikosti vstupních obrázků,
2. měřítko pro velikost obrázků pro síť která detekuje třídu a umístění. Měřítko je určeno pro snadnou změnu velikosti vstupních obrázků,
3. typ výše vstupního úhlu pro trénování. Pokud bude špatně zadaný typ tak se program po startu zeptá uživatele na zadání správného úhlu,

4. počet cyklů pro trénování obou síťí.

Ve složkách jsou umístěny většinou soubory po dvojicích, a to z důvodu testování dvou typů neuronových sítí pro detekci úhlu. V implementační části bude popsán vždy jeden soubor a to soubor s implementací založenou na [4], která byla upravena pro potřeby vlastního datasetu. První implementace sítí, pro soubory s číslem 1 a 3, která byla použita má následující strukturu, kde síť 1 je síť která se použije na detekci úhlu a síť 2 se použije na detekci umístění a třídy objektu. Síť 2 je založena na [15]. Webový článek ze kterého bylo čerpáno byl bohužel v době psaní práce smazán publikovatelem. Síť 2 využívá vlastní loss funkci pro výpočet IOU. V prvním kroku se vypočítá MSE předpovězeného a správného ohraničujícího rámce a v dalším kroku se vypočítá loss funkce 1- IOU. Matematicky lze loss funkci zapsat jako:

$$L(y, y') = MSE(y, y') + (1 - IOU(y, y'))$$

IOU je také použito jako validační metrika.



Obrázek 15: Struktura dvou neuronových sítí pro první implementaci

Soubor s neuvedeným číslem označuje první neuronovou síť pro detekci úhlu a soubory končící číslem 3 odkazují na úpravu první neuronové sítě, která navíc využívá vytváření výřezků. Základní princip zpracovávání je:

1. načtení obrázků,
2. trénování, nebo načtení vah sítě pro detekci objektů,
3. spuštění detekce objektu,

4. trénování, nebo načtení vah pro detekci úhlu,
5. podle odhadované třídy objektu se zavolá odpovídající natrénovaná strana a u třetí implementace se také vytvoří výřezek,
6. zobrazení výsledků.

Druhá implementace založená na [4] obsahuje také dvojice souborů. Značené číslem 2 a 4, kde 4 je také modifikace s vytvářením výřezků. Pro detekci používají YOLO síť a pro detekci úhlu používají Darknet-19. Tyto neuronové sítě jsem použil a upravil. V souboru je 4 je Darknetová síť zmenšena protože byly zmenšeny vstupní obrázky a síť nemohla dále zmenšovat vstupní obrázek. Síť končí na řádku 18. V souboru 2 je Darknetová síť použita celá, protože to umožnila velikost vstupního obrázku. YOLO implementace jsou použity až po řádek 15.

Tabulka 1: Struktura Darknet-19 [15]

Type	Filters	Size	Stride
Convolutional	32	3 × 3	
Maxpool		2 × 2	2
Convolutional	64	3 × 3	
Maxpool		2 × 2	2
Convolutional	128	3 × 3	
Convolutional	64	1 × 1	
Convolutional	128	3 × 3	
Maxpool		2 × 2	2
Convolutional	256	3 × 3	
Convolutional	128	1 × 1	
Convolutional	256	3 × 3	
Maxpool		2 × 2	2
Convolutional	512	3 × 3	
Convolutional	256	1 × 1	
Convolutional	512	3 × 3	
Convolutional	256	1 × 1	
Convolutional	512	3 × 3	
Maxpool		2 × 2	2
Convolutional	1024	3 × 3	
Convolutional	512	1 × 1	
Convolutional	1024	3 × 3	
Convolutional	512	1 × 1	
Convolutional	1024	3 × 3	
Convolutional	1000	1 × 1	
Avgpool		Global	
softmax			

Tabulka 2: Struktura YOLO-tiny [16]

Type	Filters	Size	Stride
Convolutional	16	3 × 3	1
Maxpool		2 × 2	2
Convolutional	32	3 × 3	1
Maxpool		2 × 2	2
Convolutional	64	3 × 3	1
Maxpool		2 × 2	2
Convolutional	128	3 × 3	1
Maxpool		2 × 2	2
Convolutional	256	3 × 3	1
Maxpool		2 × 2	2
Convolutional	512	3 × 3	1
Maxpool		2 × 2	1
Convolutional	1024	3 × 3	1
Convolutional	256	1 × 1	1
Convolutional	512	3 × 3	1
Convolutional	255	1 × 1	1
YOLO			
Route 13			
Convolutional	128	1 × 1	1
Up-sampling		2 × 2	1
Route 19 8			
Convolutional	256	3 × 3	1
Convolutional	255	1 × 1	1
YOLO			

Upravené neuronové jsou zde vypísané jako zdrojový kód podle implementace ze souboru 2. Síť Darknet-19 je upravena do následující podoby:

```
keras.layers.Conv2D(32,kernel_size=(3,3),strides=1,input_shape=(int(height/
essentials.scale_angle), int(width/essentials.scale_angle),3)),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(64, kernel_size=(3, 3), strides=1 ),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(128, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(64, kernel_size=(1, 1), strides=1),
keras.layers.Conv2D(128, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(256, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(128, kernel_size=(1, 1), strides=1),
keras.layers.Conv2D(256, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(512, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(256, kernel_size=(1, 1), strides=1),
```

```
keras.layers.Conv2D(512, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(256, kernel_size=(1, 1), strides=1),
keras.layers.Conv2D(512, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(1024, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(512, kernel_size=(1, 1), strides=1),
keras.layers.Conv2D(1024, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(512, kernel_size=(1, 1), strides=1),
keras.layers.Conv2D(1024, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(1000, kernel_size=(1, 1), strides=1),
keras.layers.AveragePooling2D(pool_size=(2, 2)) ,
keras.layers.Flatten() ,
keras.layers.Dense( 1280 ) ,
keras.layers.Dense( 640 ) ,
keras.layers.Dense( 480 ) ,
keras.layers.Dense( 120 ) ,
keras.layers.Dense( 60 ) ,
keras.layers.Dense( 1 )
```

YOLO je upravená do následující podoby:

```
keras.layers.Conv2D(16,kernel_size=(3,3),strides=1,input_shape=(int(height/
essentials.scale_box), int(width/essentials.scale_box), 3)),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(32, kernel_size=(3, 3), strides=1 ),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2),
keras.layers.Conv2D(64, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2),
keras.layers.Conv2D(128, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2),
keras.layers.Conv2D(256, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2 ),
keras.layers.Conv2D(512, kernel_size=(3, 3), strides=1),
keras.layers.MaxPooling2D( pool_size=( 2 , 2 ), strides=2),
keras.layers.Conv2D(1024, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(256, kernel_size=(1, 1), strides=1),
keras.layers.Conv2D(512, kernel_size=(3, 3), strides=1),
keras.layers.Conv2D(255, kernel_size=(1, 1), strides=1),
keras.layers.Flatten() ,
keras.layers.Dense( 1280 ) ,
keras.layers.Dense( 640 ) ,
keras.layers.Dense( 480 ) ,
keras.layers.Dense( 120 ) ,
keras.layers.Dense( 60 ) ,
keras.layers.Dense( 5 )
```

Soubor 4 odpovídá logice a struktuře v [4]. Načte se obrázek, zjistí se umístění objektu, vytvoří se výřez nalezené plochy a výřez se pošle do detektoru úhlu. Rozdíl je že implementace v diplomové práci výřez neotáčí v posledním kroku.

4.3.1 Soubor s celým trénováním

Jedná se o soubory train. Tento soubor v sobě kombinuje trénování úhlu a trénování detektoru, struktura je založena na principu YOLO Barcode algoritmu. V prvním kroku proběhne detekce objektu a následně podle predikované třídy se zavolá odpovídající natrénovaná strana úhlu. Po potřebných importech a načtení dat z konfiguračního souboru se načtou data pro trénování detektoru umístění a třídy objektu a reálné testovací obrázky.

Následuje třída pro trénování úhlů úplně stejně jako v předchozím souboru, tuto třídu také trénuji stejně pro každou ze 4 stran Alienátora. Plynně navazuje část, která se stará o trénování detektoru. V obou těchto částech je určen model sítě a kody pro jeho trénování a uložení.

Spustí se trénování detektoru umístění objektu a třídy a následně se natrénují úhly a spustí se testování, které projde všechny testovací obrázky ve složce real a výsledné detekce se uloží do obrázku do odpovídající složky detections. Podle předpovězené třídy a tím strany objektu bude načten odpovídající úhlový soubor.

4.3.2 Soubor s načítáním a společnými daty

Jedná se o soubory prediction. Tento soubor netrénuje, ale pouze načítá natrénované modely a spouští testovaní a generování výsledků. Váhy uložené z předchozího souboru se použijí pro tento soubor, jenž vznikl jako snaha o opětovné spuštění moselů bez potřeby spuštění trénování.

Začátek souboru tvoří importy všech použitých knihoven, načtení konfiguračního souboru a reálných testovacích dat. Načte se model detektoru umístění objektu a třídy vytvoří se predikce, které se zpracují do výsledných obrázků.

Dalším důležitým souborem je soubor essentials, který obsahuje části společné pro všechny ostatní soubory. Obsahem tohoto souboru jsou data načtená z konfiguračního souboru, názvy tříd, metody použité při trénování, metoda pro úpravu dat, metody pro načtení dat pro trénování a testování a na konci souboru se nachází dvě metody pro vygenerování grafů, které se uloží do odpovídající složky detections.

Kromě výše uvedených .py souborů jsou ve složce umístěné složky s obrázky a uložené natrénované modely. Uvedu zde krátký popis zbývajících částí:

- složka alienator obsahuje celý dataset pro trénování detektoru;
- složka configs obsahuje konfigurační soubory pro trénování a zpracování obrázkových dat;
- složky detections obsahují výsledky z trénování celých detektorů. Číslo značí použitou síť a písmeno pořadí;
- složka real obsahuje reálné obrázky, které se používají pro testování výsledků.

4.4 Tensorflow 2.0 YOLO

V následující části je popsána implementace YOLO v3. Tato implementace byla použita a upravena pro účel porovnání výsledků z konvoluční neuronové sítě, které nedosahovaly takových výsledků, jaké by byly ideální. Tato implementace je k dispozici na webu GitHubu od autora penny4860 [3].

Tato implementace využívá novou funkci TensorFlow 2.0 dychtivého spuštění. Pro spuštění je potřeba splnit následující předpoklady:

- Python 3.6 a vyšší;
- TensorFlow 2.0.0-beta1;
- a další knihovny jako OpenCV, Pillow, pytest-cov, codecov, matplotlib a další. Celý seznam je na GitHubu autora a v návodovém souboru.

V projektu se nachází složky a zdrojové kódy. První složka configs obsahuje konfigurační soubory pro trénování, které jsou uložené ve formátu .json, zároveň se do této složky ukládají uložené natrénované váhy. Konfigurační soubor obsahuje údaje o kotvách, názvu tříd, místa uložení natrénovaných vah, rozměry a umístění obrázků a anotací. Uvedu zde konfigurační soubor, který se používá pro trénování. Ukládají se zde také konfigurační soubory z předchozí části, které se používají do trénování detektoru úhlu. Konfigurační soubor pro YOLO má následující strukturu:

```
{"model" : {
"anchors":[73,112,113,110,93,134,79,172,142,109,119,139,103,162,145,121,135
,133],
"labels":["front Alienator","right Alienator","back Alienator","left Aliena
tor"],
"net_size":352},
"pretrained":{
"keras_format":"configs/alienator/weights.h5",
"darknet_format":"yolov3.weights",
"angle_format1":"configs/alienator/model_front.h5",
"angle_format2":"configs/alienator/model_right.h5",
"angle_format3":"configs/alienator/model_back.h5",
"angle_format4":"configs/alienator/model_left.h5" },
"train":{
"min_size":288,
"max_size":416,
"num_epoch":100,
"train_image_folder":"tests/dataset/alienator/imgs",
"train_annot_folder":"tests/dataset/alienator/anns",
"valid_image_folder":"tests/dataset/alienatoT/imgs",
"valid_annot_folder":"tests/dataset/alienatoT/anns",
"batch_size":4,
"learning_rate":1e-4,
"save_folder":"configs/alienator",
"jitter":false }}
```

Složka imgs obsahuje dobře přístupné obrázky pro predikci obrázků po natrénování. Do této složky se vkládá testovací dataset pro snadnější přístup z hlavní složky.

Složka tests obsahuje složku dataset, ve které jsou umístěny všechny datasety pro trénování a validaci, dále zde byly umístěny testovací a kontrolní soubory.

Složka yolo obsahuje několik dalších podsložek a velké množství .py souborů, které tvoří hlavní část funkčnosti Yola. Nalezneme zde soubor pro zpracování konfiguračního .json souboru, trénovací skript a evaluační skript, ve složkách jsou soubory pro zpracování datasetu, výpočet loss funkce, samotné části neuronové sítě jako váhy, kostru neuronové sítě a zobrazovací funkce ohraničujícího rámce. Počet souborů je zde velký, ale menší než v Darknetu. Tato skutečnost zajišťuje lepší porozumění kódu a následné úpravy.

Jelikož tato implementace YOLa využívá pro anotaci .xml soubory, tak nebylo potřeba velkých úprav existujících označovacích souborů, ale na druhou stranu bylo potřeba upravit konfigurační soubor. V tomto souboru bylo nutné upravit anchors a labels. Správné hodnoty pro kotvy v datasetu se vypočítaly s pomocí Darknetové implentace YOLa, jehož detektor nabízí funkci calc_anchors. Do labels bylo potřeba vložit nové názvy tříd. Následně bylo nutné přidat odkazy na natrénované váhové soubory pro detektor tříd, ty jsou pojmenované jako angle_format. V posledním kroku bylo nutné upravit cesty k datasetům a místu pro uložení natrénovaných vah.

Trénování úhlu je řešeno separátně s trénováním YOLa. Úhly a detektor se natrénují samostatně a až během evaluace se jejich výsledky spojí do výsledného zobrazení.

Prvním souborem se zdrojovým kódem, který byl upraven, je pred.py, který se stará o načtení jednoho testovacího obrázku, jeho otestování na načtené síti a vytvoření výsledné predikce. Ve funkci main se vytvoří YOLO model a zpracují se načtené váhy, následně dojde k načtení obrázku a zde se ukážou první úpravy. Načtený obrázek se poslal do funkce make_array, aby došlo k jeho zabalení do pole a zavolala se funkce load_label pro vytvoření pole, které obsahuje informace o obrázku. Poté se spustí detektor a tím dostaneme predikce.

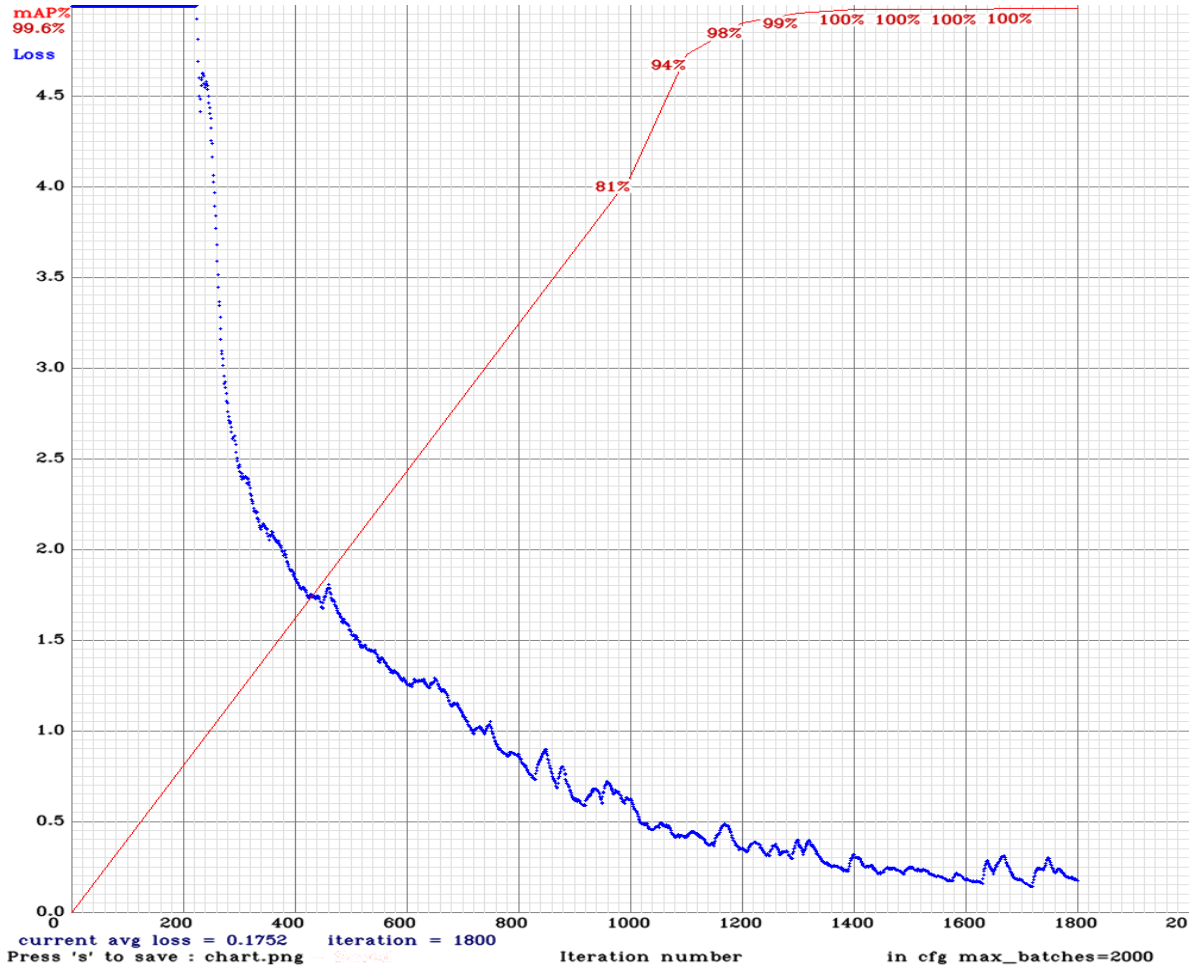
Při testování programu se u poslední chybové třídy stávalo, že YOLO mělo prázdné výstupy, které znemožnily zobrazení výsledků, proto je přidán kód, který prázdný výstup nahradí standardním chybovým výstupem.

Jelikož implementace YOLa obsahuje více výsledků z hlediska ohraničujících rámců, pravděpodobností a názvů tříd, byl přidán malý kus kódů pro nalezení maximální pravděpodobnosti, podle níž se vybere nejlepší název třídy a ohraničující rámce, které se použijí pro závěrečné zobrazení výsledků. Při posledním trénování se stávalo že chybová třída byla u každého obrázku s nejvyšší pravděpodobností, proto je ve zdrojovém kodu upravená verze minulého kodu, která vybere třídu a ohraničující rámec s druhou nejvyšší pravděpodobností, pokud je ve výsledném poli více hodnot. Tento kod je v odevzdané části zakomentován. Následuje samotné zobrazení výsledků do obrázku a uložení obrázku do složky jako detection.png. V souboru pred.py byly použity upravené funkce také z jiného souboru: config.py, v tomto souboru byla přidána funkce k načtení natrénovaných úhlů pro každou ze čtyř stran. Trénování těchto úhlů probíhá v souboru train_angle.py, který je založený na z části implementace v 4.3.1, která se stará o trénování detektoru úhlu. Ve složce se nachází upravený soubor essentials.py, který obsahuje všechna společná data pro train_angle.py

V této podkapitole byly popsány úpravy, které byly provedeny v implementaci YOLO, aby bylo dosaženo požadované funkčnosti. Celkově tato kapitole nastínila strukturu označovací programu a Darknetové implementace YOLO. Větší důraz byl kladen na implementace v TensorFlow, které se nachází na disku ve složce aplikace a aplikace-YOLO. U implementace byla popsána struktura obou implementací.

5. Výsledky

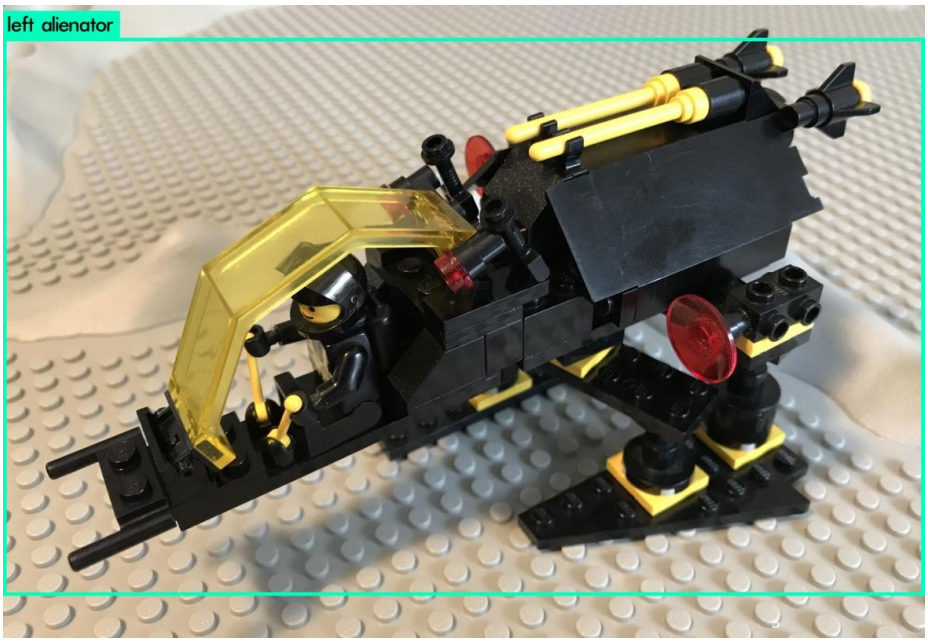
Poslední kapitola této diplomové práce se bude zabývat výsledky z darknetové implementace YOLa a také z Pythonové implementace vlastního řešení a upraveného YOLa. Jak bylo zmíněno v minulých kapitolách, implementace Darknet byla použita jen na samotnou detekci objektů a odhad úhlu se v ní nenacházel. Trénování bylo prováděno na notebooku Lenovo Y510p, který disponuje 2 grafickými kartami NVIDIA GeFroce GT 755M v SLI konfiguraci s velikostí paměti 2 GB pro každou grafickou kartu. Při trénování byla aktivní jedna grafická karta. Doba trénování YOLO-tiny byla šest hodin nepřerušeného trénování. Trénování základního YOLa nebylo možné kvůli nedostatku paměti na grafické kartě. Průběh trénování je vidět na následujícím grafu. Všechny výsledky jsou s horizontálním úhlem (úhlem natočení).



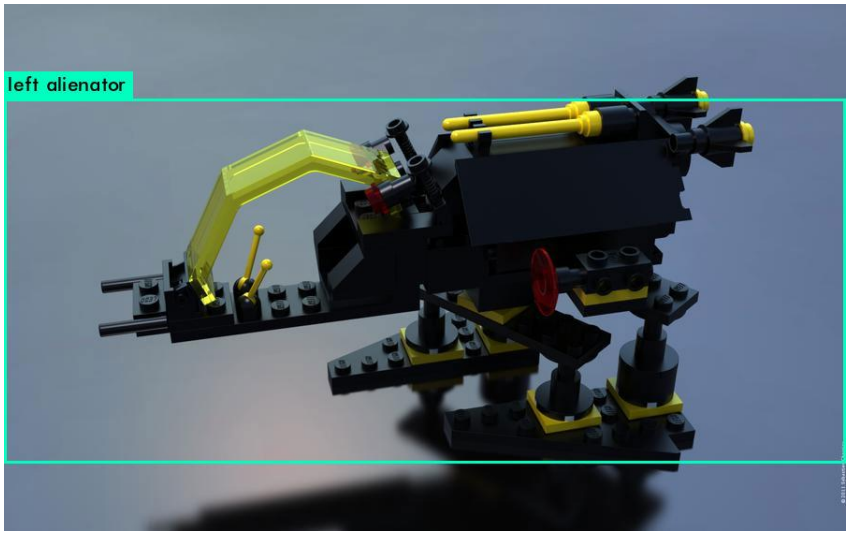
Obrázek 16: Ukázka procesu trénování Darknetu

Toto trénování bylo spuštěno celkem čtyřikrát, vždy v konfiguračním souboru byly provedeny drobné úpravy. Podle různých trénování a úprav lze dojít k závěru že základní YOLO-tiny je nejlepší. Menší verze byla rychlejší a dosáhla podobné přesnosti jako základní verze, ale její IOU má vyšší hodnotu než základní verze. Větší verze by nejspíše dosáhla vyšší přesnosti, ale její trénování je velmi pomalé. Zpomalení je způsobeno malým zvětšením sítě a nutností zmenšit množství paralelně běžících vláken kvůli paměťovým možnostem grafické karty. Struktura YOLOv3-tiny je popsána v tabulce 2.

Pro pochopení provedených úprav je důležitá pozice YOLO vrstev. Základní YOLO-tiny má 2 YOLO vrstvy. Modifikace 2 z obrázku 16 byla menší síť, která obsahovala všechno do vrstvy 16. Obsahovala 1 YOLO vrstvu. Modifikace 2 obsahovala úpravu mezi vrstvou 21 a 22. Mezi tyto vrstvy byla přidána vrstva Up-sampling, Convolutional a Convolutional. Dále zde uvedu 2 výsledky z testování algoritmu na obrázcích. První odhad má přesnost 100% a druhý 98% procent. Odhady byly většinou velmi dobré, nad 80%. Čím větší byl Alienátor na zadaném obrázku tím větší byla přesnost.



Obrázek 17**:** Ukázka výsledku Darknet YOLO detekce



Obrázek 18: Ukázka výsledku Darknet YOLO detekce

Je vidět, že třídy jsou zobrazeny správně. Dále je vidět, že třídy jsou zobrazeny správně a ohraničující rámec plně obsahuje hledaný objekt. Výsledné IOU bude vyšší než 95%. Jediná drobná část která chybí v ohraničujícím rámci je kostka zadního raketového motoru. Pokud jsou hledané objekty menší v zadaném obrázku tak detektor ukazuje menší jistotu odhadu. Obvyklý odhad přesnoti je nad 90%. Celkově jsou výsledky pro YOLOv3 velmi přesné.

5.1 Výsledky z první implementace

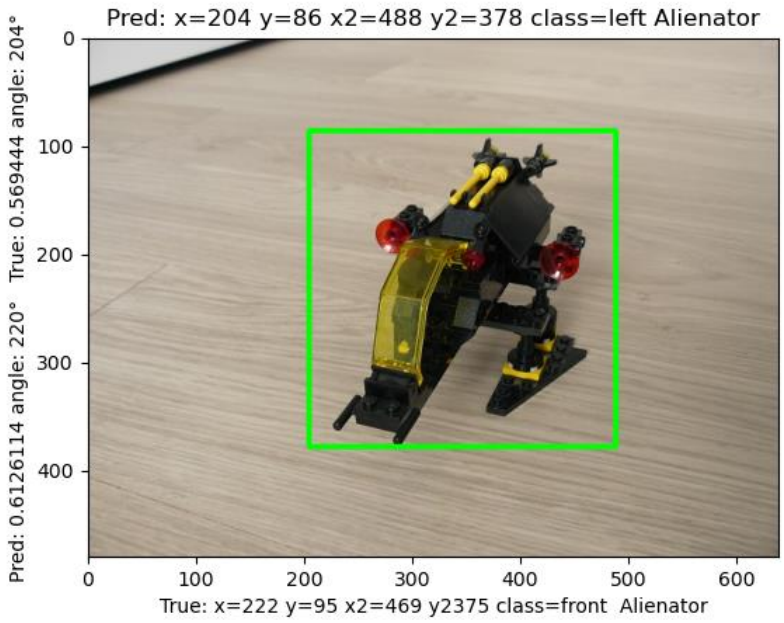
To je důležitá část diplomové práce. Přestože jsou popsané dva typy souborů, výsledky budou popsány jen jedny, protože oba druhy souborů ukazují stejný výsledek. Celkem je vytvořených 66 výstupů pro každou natrénovanou třídu na zadaném počtu epoch. Ve výstupu jsou v horní liště nad obrázkem vypsané hodnoty vygenerované první konvoluční neuronovou sítí. Na výstupu je x, y levého horního bodu ohraničujícího rámce a x, y hodnoty pravého dolního rohu ohraničujícího rámce doplněné o číslo, které je převedeno na název třídy. Názvy těchto tříd jsou:

- left Alienator
- back Alienator
- right Alienator
- front Alienator
- error

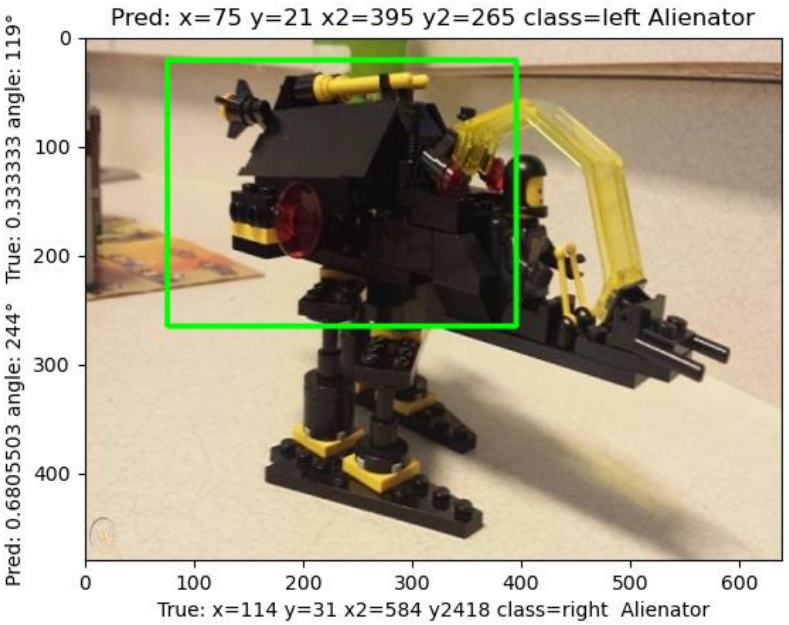
Pro soubor 1 a 3 byla neuronová síť pro detekci úhlu natrénována na 1000 epoch a konvoluční neuronová síť pro detekci ohraničujícího rámce a třídy na 200 epoch. Trénování menší sítě trvalo v řádu minut a trénování druhé sítě trvalo v řádu hodin. Natrénování čtvrtého souboru na 200 epoch trvalo 18 hodin. Tento čas se bude lišit na počítačích s jiným hardwarovým výkon a při při úpravě velikost batch_size. Na pracovním počítači byla hodnota batch_size nastavená na hodnotu 4, aby nedošlo k přeplnění paměti RAM. Pro soubor 2 a 4 byla konvoluční neuronová síť pro detekci úhlu natrénována na 200 epoch a konvoluční neuronová síť pro detekci ohraničujícího rámce a třídy na 200 epoch. Trénování menší sítě obou trvalo v řádu hodin, ale trénování úhlu bylo kratší kvůli menšímu datasetu. Proces trénování probíhal na procesoru, protože TensorFlow potřebuje pro trénování na grafické kartě compute capability 3,5 a vyšší, ale grafické karty na pracovním notebooku mají compute capability 3,0. Při testovacím trénování byl rozdíl proti závěrečné aplikaci v práci se stranami. Testování nemělo rozdělené strany, ale všechny strany byly spojené, zatímco v závěrečné aplikaci jsou strany rozdělené.

Pod obrázkem se nacházejí data ve stejném formátu, ale ta jsou načtena z anotovacího souboru. Na levé straně od obrázku se nachází informace od úhlu, kdy dole je úhel natočení odhadovaný neuronovou sítí a nahoře jsou hodnoty z anotovacího souboru. Úhel je vyjádřený jako normalizované číslo s desetinnou čárkou představující $\frac{X}{360}$ úhlu, které je ve výstupu zobrazeno také ve stupních.

Při porovnání výsledků z vlastní implementace a YOLO implementace je z výsledků detekce úhlů vidět, že hodně záleželo na samotném obrázku. U obrázků s malým Alienátorem byl úhel odhadnut lépe druhou sítí, ale na snímcích, kde byl Alienátor stejně velký jako v testovací množině, vyšla lépe první síť. Při pohledu na výsledky testování hodně záleží na velikosti Alienátora v obrázku a jak je trénování nastaveno. Při trénování všech stran najednou je lepší druhá síť, ale při rozdělených stranách je lepší první síť. Druhý případ je důležitější pro práci, proto i další výsledky testů v dalších kapitolách budou založeny na výsledcích z první sítě.



Obrázek 19: Ukázka výstupu



Obrázek 20: Ukázka výstupu

Toto je jeden ukázkový výstup z detekce. Ve stejném formátu jsou uvedeny ostatní výstupy. Popisky jsou uvedeny pro snadné porovnání referenčních a předpovězených hodnot. Výstupní souřadnicová data jsou doplněna ohraničujícím rámcem v obrázku.

V další části jsou uvedeny tabulky výsledků pro starší verzi testovacího datasetu a novější verzi testovacího datasetu. Liší se množstvím a typy obrázků v nich uložených. Odevzdaná příloha pracuje s novou verzí testovacího datasetu.

Tabulka 3: Výsledky na testovacím datasetu (nový testovací dataset)

Odchylka úhlů	1. soubor 200 epoch	3. soubor 200 epoch	2. soubor 100 epoch	2. soubor 200 epoch	4. soubor 100 epoch	4. soubor 200 epoch
Odchylka 0°–10°	2	1	4	1	3	3
Odchylka 10°–30°	2	7	15	6	5	3
Odchylka 30°–50°	3	3	1	4	7	4
Odchylka 50° a více	30	21	13	21	22	21
Třída objektu						
Správná	16	17	20	13	19	19
Špatná	37	36	33	40	34	34
Ohraničující rámec						
IOU <5%	2	0	8	9	9	3
IOU >=5% & IOU <95%	17	13	17	16	17	19
IOU >=95%	36	40	30	30	29	31

Tabulka 4: Výsledky na testovacím datasetu (starý testovací dataset)

Odchylka úhlů	1. soubor 200 epoch	3. soubor 200 epoch	2. soubor 100 epoch	2. soubor 200 epoch	2. soubor 400 epoch	4. soubor 100 epoch	4. soubor 200 epoch	4. soubor 400 epoch
Odchylka 0°–10°	2	2	4	1	9	1	2	4
Odchylka 10°–30°	2	6	16	9	13	6	3	7
Odchylka 30°–50°	5	2	6	2	16	22	5	11
Odchylka 50° a více	44	20	10	18	15	26	18	20
Třída objektu								
Správná	28	17	25	13	35	28	12	23
Špatná	38	49	41	53	31	38	54	43
Ohraničující rámec								
IOU <5%	32	32	32	37	28	30	5	28
IOU >=5% & IOU <95%	8	6	11	7	8	8	30	13
IOU >=95%	26	28	23	22	30	28	31	25

Vysvětlivky:

- Odchylka: jedná se o absolutní rozdíl mezi referenčním úhlem a předpovězeným úhlem.
- Správná: předpovězená třída objektu je shodná s referenční třídou objektu.
- Špatná: předpovězená třída objektu není shodná s referenční třídou objektu.

Z výsledků na novém datasetu je vidět že nejlépe pro úhel a detekci správné třídy objektu vyšla druhá implementace trénovaná na 100 epoch. Z pohledu přesnosti vykreslené plochy vyšla nejlépe implementace 3. Další analýza je pro výsledky ze staršího datasetu.

U chybových obrázků nebyl rozptyl úhlů započítán, a u obrázků s Alienátorem, u kterých byla předpovězena chybová třída, také. U obrázků se špatně odhadovanou třídou bude rozdíl úhlů ve většině případů větší než 50°, ale byly případy kde tomu tak nabylo. Hlavně v oblasti přechodu z jedná strany na druhou. V tabulce je vidět, že většina obrázků měla velký rozptyl úhlů, většina tříd byla špatně odhadnuta a překrytí objektů bylo většinou velmi nepřesné.

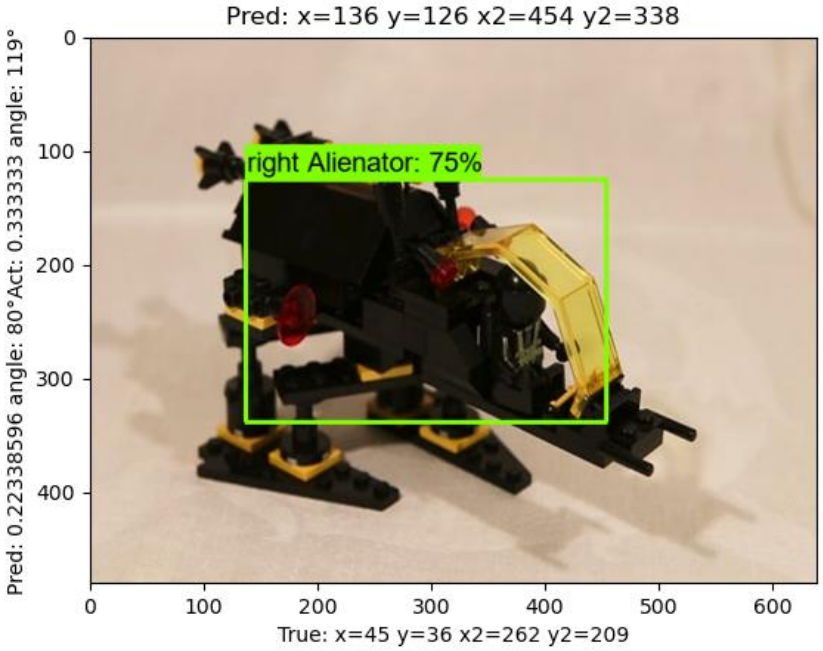
Tyto výsledky jsou z velké části dány opakováním prakticky toho stejného obrázku, velkou část mého testovacího datasetu tvoří obrázek kde je Alienátor v menším rozměru a pravé polovině obrázku. Jedná se o obrázky, které se liší vždy v jednom stupni natočení. U těchto obrázků byla u některých obrázků zajímavá změna odhadovaných tříd a zároveň ve většina případů odhad ohraničujícího rámce do levého horního rohu ve kterém se nachází část skříně. detekci části skříně nejspíše bude způsobená obrázky stolů z trénovacího datasetu a tím že Alienátor je v těchto obrázcích menčí než v ostatních. Překrytí bylo nejčastěji mimo hledaný objekt. Pro správnější detekci by pomohlo mít síť jako scale-invariant. Rozptyl úhlů se u obrázků s malým Alienátoremvelmi liší ale ve většině případů je nad 50°. Odhad úhlu pro neuronové sítě s vytvářením výřezů vždy vyhodí špatné číslo a pokud se bude blížit tak bude záležet na náhodě kvůli špatném umístění ohraničujícího rámce na části skříně v levém horním rohu.

U ostatních obrázků s větším alinátorem byl ohraničující rámec nakreslen správně nebo úplně správně na hledaném objektu. Pokud došlo ke správné lokalizaci strany, tak s velkou pravděpodobností byl úhel správně odhadnutý s rozptylem menším než 30. Z hlediska úhlů byla překvapivá situace u několika obrázků, kde přestože byla špatně odhadnuta strana a zavolal se odhad úhlu na tuto špatnou stranu, výsledek vyšel správně.

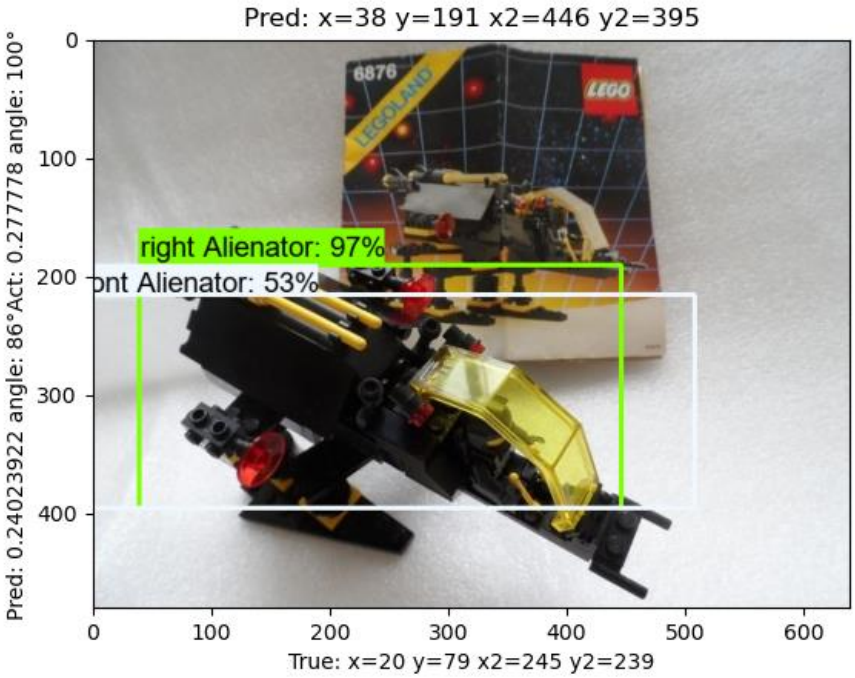
Výsledky u chybových obrázků jsou se někdy stávalo že v nich byl špatně odhadnuta přítomnost Alienátora, který se v obrázku nenachází. Toto bude nejspíše dáno velkým množstvím cybových obrázků při trénování s různými typ pozadí. Pro chybové obrázky se nepočítá úhel, třída je určena ve většině případů správně a obrázek má jen malý ohraničující rámec v levém horním rohu, nebo na místě špatného objektu. Výsledek lze označit za dobrý, ale ne zázračný.

5.2 Výsledky z druhé implementace

Tato implementace by podle předpokladu měla dosáhnout lepšího odhadu ohraničujícího rámce a třídy objektu. Formát výstupu je stejný, jedná se o obrázek s popisky po stranách. Nad obrázkem jsou umístěny odhadované hodnoty ohraničujícího rámce a pod obrázkem pravdivé hodnoty ohraničujícího rámce. Třída objektu je napsána v obrázku. Nalevo od obrázku jsou napsány údaje o úhlu. Celkově je výpis velmi podobný formátu z minulé implementace.



Obrázek 21: Ukázka výstupu YOLO



Obrázek 22: Ukázka výstupu YOLO

V další části jsou uvedeny tabulky výsledků pro starší verzi testovacího datasetu a novější verzi testovacího datasetu. Liší se množstvím a typy obrázků v nich uložených. Odevzdaná příloha pracuje s novou verzí testovacího datasetu.

Tabulka 5: Výsledky YOLa na testovacím datasetu (nový testovací dataset)

Odchylka úhlů	soubor 1	soubor 3	soubor 2-1	soubor 2-2	soubor 2-3	soubor 4-1	soubor 4-2	soubor 4-3
Odchylka 0°–10°	3	1	4	10	9	1	4	4
Odchylka 10°–30°	7	12	9	12	11	10	14	14
Odchylka 30°–50°	7	10	5	6	7	6	8	7
Odchylka 50° a více	12	6	1	1	2	2	3	5
Třída objektu								
Správná	42	41	35	43	44	35	43	44
Špatná	12	13	19	11	10	19	11	10
Ohraničující rámec								
IOU <5%	9	8	19	5	9	19	4	9
IOU >=5% & IOU <95%	7	7	3	4	4	3	6	5
IOU >=95%	38	39	32	45	41	32	44	40

Tabulka 6: Výsledky YOLa na testovacím datasetu (starý testovací dataset)

Odchylka úhlů	soubor 1	soubor 3	soubor 2-1	soubor 2-2	soubor 2-3	soubor 4-1	soubor 4-2	soubor 4-3
Odchylka 0°–10°	4	2	16	19	20	1	4	3
Odchylka 10°–30°	13	16	11	18	10	24	24	19
Odchylka 30°–50°	12	12	4	10	15	5	14	17
Odchylka 50° a více	22	21	2	5	5	2	11	11
Třída objektu								
Správná	60	59	45	58	59	47	60	59
Špatná	6	7	21	8	7	19	6	7
Ohraničující rámec								
IOU <5%	7	7	21	5	7	19	6	7
IOU >=5% & IOU <95%	5	5	2	4	5	0	6	5
IOU >=95%	54	54	43	57	54	47	54	54

Vysvětlivky:

- soubor X-1: používá pro natrénování YOLa stažené YOLOv3 váhy. Soubor 1 a 3 mají použité pro výsledky také nově natrénované váhy.
- soubor X-2: používá YOLO váhy natrénované na starém datasetu
- soubor X-3: používáYOLO váhy natrénované na starém datasetu, které byly přetrénovány novým datasetem.
- Ostatní vysvětlivky jsou shodné s vysvětlivkami první tabulky.

Z výsledků na novém datasetu je vidět že nejlépe pro úhel vyšla druhá implementace s váhami trénovanými na starém datasetu. Odhad správné třídy mají nejlepší implementace s přetrénovanými starými vahami a přesnost odhadu místa mají váhy trénované na starém datasetu. Další analýza je pro výsledky ze staršího datasetu.

Jako u předchozích obrázků nebyl u chybových obrázků počítán rozptyl úhlů a u špatně předpovězených chybových obrázků také. Z obrázků je vidět, že třída objektu je ve většině případů odhadnuta správně. Většinou je zobrazena jedna třída, ale jsou i obrázky, kde je tříd odhadnuto více pro různé strany. Přesnost odhadu je u obrázků s velkým Alienátorem nad 80 %. V tabulce je vidět že většina obrázků třídy byla správně odhadnuta, překrytí objektů bylo až na výjimky úplně přesné a odhad úhlu byl nejlepší pro druhou síť trénovanou na starými váhami a přetrénovanými váhami.

U vah trénovaných na strarém datasetu je vidět, že jsou mnohem citlivější a dávají více odhadů než přetrénované váhy a nově natrénované váhy. Starší váhy ve druhém souboru dosahují nejlepší výsledky pro kvalitu odhadu úhlu, správnosti odhadnuté třídy a správnosti odhadu úhlu. Ohraničující rámce ve většině případů dobře, největší nedostatky se vyskytovaly u nově natrénovaných vah, které u některých obrázků měly problém s detekcí Alienátora který byl v obrázku umístěný. Překvapivé zjištění bylo že menšího Alienátora dokázal vyhledat, ale většího v jiném obrázku ne. Výsledek je lepší z pohledu přesnosti ohraničujícího rámce a z pohledu úhlu jde také o zlepšení v porovnání s předchozí implementací. Je v překvapivé že 4. soubor, který odpovídá architektuře inspirativního YOLO Barcode projektu, nemá největší přesnost, ale jako nejlepší se ukázala implementace 2, která neobsahuje vytváření výřezů odhadovaného umístění objektu. Výsledky upravené YOLO implmenetace dosahují lepších výsledků než vlastní implementace.

6. Závěr

Cílem práce bylo vytvořit a otestovat aplikaci na vyhledávání objektů v obraze a určit úhel natočení hledaného objektu. Aplikace umí vizuálně ukázat odhadované umístění objektu a odhadnout třídu daného objektu. V dalším kroku se na základě nalezené pozice provede odhad úhlu. Pro zpracování úlohy je možné využít framework jako TensorFlow, YOLO Darknet nebo PyTorch. Z těchto frameworků byl vybrán TensorFlow ve verzi 2.0.

Vývoj aplikace probíhal postupně zjištěním informací o zadaném tématu. Po konzultaci byly vybrány neuronové sítě UnitBox a YOLO na testování. Testoval jsem Darknetovou implementaci neuronové sítě YOLO, která měla výborné výsledky hledání, ale špatně se framework upravoval pro detekci úhlu. V další fázi se přešlo na TensorFlow a v této knihovně jsem implementoval řešení a následně upravil existující implementaci YOLa v této knihovně. Před odevzdáním práce jsem testoval implementaci RotationNetu.

V textu diplomové práce jsem popsal syntetické obrazy, neuronové sítě, použité knihovny, aplikaci a její výsledky. U syntetických obrazů a neuronových sítí jsem se zaměřil na části, které jsou důležité pro diplomovou práci. Podařilo se mi vytvořit neuronovou síť a upravit stávající neuronovou síť tak, aby uměla najít pozici objektu a odhadla jeho úhel. Síť jsem natrénoval na datasetu ze syntetických obrázků a otestována byla na testovacím datasetu reálných obrázků. První implementace by bylo ideální plně doladit, aby měla co nejlepší výsledky. Omezení první implementace je, že není trénována pro různá rozlišení vstupního obrázku. Dalším častým problémem byl špatný odhad třídy objektu, který v dalším kroku velmi ovlivňoval implementace které vytvářely výřez odhadovaného umístění objektu. Po rozšíření trénovacího datasetu je vidět větší nepřesnost výsledků a zhoršená kvalita odhadů.

Z výsledků vyšla nejlépe implementace YOLa ze souboru číslo 2, která měla největší přesnost nalezení správného umístění objektu a jeho třídy a zároveň nejlepší odhad úhlu pod kterým objekt vidíme. Lepšímu odhadu úhlu pomáhá přesnější odhad třídy, podle kterého se volá odpovídající natrénovaná síť.

Strukturou pro vytvoření sítě jsem se inspiroval ve zdroji [4]. Částečně se jedná o originální řešení, protože aplikuje konvoluční neuronové sítě a obrázky s 3D objekty. YOLO a ostatní konvoluční neuronové sítě se zabývají detekcí úhlu 2D objektu v obraze, například zmíněný čárový kód. Tyto sítě jsou v diplomové práci použité na nalezení orientace 3D objektu v obraze. Pro tento úkol jsou určeny neuronové sítě jako například RotationNet a podobné.

Podařilo se mi splnit zadání práce a implementace s upraveným YOLO algoritmem dosahuje celkem dobrých výsledků. Práce má prostor pro doladění a vyzkoušení dalších implementací detektoru objektů a úhlů.

Literatura

[1] RAJPURA, P. S. Object Detection Using Deep CNNs Trained on Synthetic Images. *SemanticScolar* [online]. Seattle, 2015, 2017 [cit. 2020-03-14]. Dostupné z: https://www.semanticscholar.org/paper/Object-Detection-Using-Deep-CNNs-Trained-on-Images-Rajpura-Hegde/278854f6efe41b253fb4564b71e4a87d224b0cc0

[2] SCHRAML, Dominik. Physically based synthetic image generation for machine learning: a review of pertinent literature. *SPIE.DIGITAL LIBRARY* [online]. Bellingham, 1995, 2019 [cit. 2020-03-14]. Dostupné z: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11144/111440J/Physically-based-synthetic-image-generation-for-machine-learning--a/10.1117/12.2533485.full?SSO=1

[3] HOANG LE, Tuan Anh. Using synthetic data to train neural networks is model-based reasoning. *ResearchGate* [online]. Boston, 2008, 2017 [cit. 2020-03-14]. Dostupné z: https://www.researchgate.net/publication/318332084_Using_synthetic_data_to_train_neural_networks_is_model-based_reasoning

[4] HANSEN, Daniel Kold. Real-Time Barcode Detection and Classification using Deep Learning. *ResearchGate* [online]. Boston, 2008, 2017 [cit. 2020-03-14]. Dostupné z: https://www.researchgate.net/publication/321065123_Real-Time_Barcode_Detection_and_Classification_using_Deep_Learning

[5] LIU, Lei. Learning a Rotation Invariant Detector with Rotatable Bounding Box. *ResearchGate* [online]. Boston, 2008, 2017 [cit. 2020-03-14]. Dostupné z: https://www.researchgate.net/publication/321329814_Learning_a_Rotation_Invariant_Detector_with_Rotatable_Bounding_Box

[6] SACHAN, Ankit. Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD by Ankit Sachan. *CV-Tricks* [online]. 2017 [cit. 2020-03-14]. Dostupné z: https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[7] *TensorFlow* [online]. USA, 2017 [cit. 2020-03-14]. Dostupné z: https://www.tensorflow.org/

[8] SONG, Chen. HybridPose: 6D Object Pose Estimation under Hybrid Representations. *Cornell University* [online]. Ithaca, New York [cit. 2020-03-14]. Dostupné z: https://arxiv.org/abs/2001.01869

[9] XIANG, Yu. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *Cornell University* [online]. Ithaca, New York [cit. 2020-03-14]. Dostupné z: https://arxiv.org/abs/1711.00199

[10] VENKAT, Suraj. Best Image Labeling Tools For Computer Vision. *Cornell University* [online]. Ithaca, New York [cit. 2020-03-14]. Dostupné z: https://medium.com/tektorch-ai/best-image-labeling-tools-for-computer-vision-393e256be0a0

[11] KANEZAKI, Asako. RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints. *Cornell University* [online]. Ithaca, New York [cit. 2020-03-14]. Dostupné z: https://arxiv.org/abs/1603.06208

[12] TOSHEV, Alexander. DeepPose: Human Pose Estimation via Deep Neural Networks. *Cornell University* [online]. Ithaca, New York [cit. 2020-03-14]. Dostupné z: https://arxiv.org/abs/1312.4659

[13] Neuronové sítě [online]. Brno [cit. 2020-05-02]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21471

[14] Co je to neuron ? *Jamesove stránky* [online]. 2008 [cit. 2020-05-02]. Dostupné z: https://survivebatchanimals.7x.cz/rubriky/co-je-to-neuron

[15] *Medium* [online]. [cit. 2020-06-02]. Dostupné z: https://towardsdatascience.com/getting-started-with-bounding-box-regression-in-tensorflow-743e22d0ccb3

[16] ZAKHAROV, Sergey. *DPOD: 6D Pose Object Detector and Refiner* [online]. Technical University of Munich, 2019 [cit. 2020-06-02]. Dostupné z: https://arxiv.org/pdf/1902.11020v3.pdf

[17] KEHL, Wadim. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. *Cornell University* [online]. Technical University of Munich, 2017 [cit. 2020-06-02]. Dostupné z: https://arxiv.org/pdf/1711.10006.pdf

[18] CAO, Zhe. OpenPose: Realtime Multi-Person 2D PoseEstimation using Part Affinity Fields. *Cornell University* [online]. 2019 [cit. 2020-06-02]. Dostupné z: https://arxiv.org/pdf/1812.08008.pdf

[19] Datasets. *BOP: Benchmark for 6D Object Pose Estimation* [online]. [cit. 2020-07-02]. Dostupné z: https://bop.felk.cvut.cz/datasets/

Seznam zdrojů obrázků

Obrázek 1	https://www.mentem.cz/blog/neuron/
Obrázek 2	https://towardsdatascience.com/the-perceptron-3af34c84838c?gi=9e8a4e2b32df
Obrázek 3	https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
Obrázek 4	https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
Obrázek 5	https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
Obrázek 6	https://github.com/liulei01/DRBox
Obrázek 7	https://vbn.aau.dk/ws/portalfiles/portal/261392843/yolo_barcode.pdf
Obrázek 8	https://www.semanticscholar.org/paper/PoseCNN%3A-A-Convolutional-Neural-Network-for-6D-Pose-Xiang-Schmidt/1c7e078611c9df412e6eb3a356f31a0da0c1f99c/figure/1
Obrázek 9	https://www.semanticscholar.org/paper/PoseCNN%3A-A-Convolutional-Neural-Network-for-6D-Pose-Xiang-Schmidt/1c7e078611c9df412e6eb3a356f31a0da0c1f99c/figure/1
Obrázek 10	http://host.robots.ox.ac.uk/pascal/VOC/voc2012/
Obrázek 11	http://cocodataset.org/#explore
Obrázek 14	https://upload.wikimedia.org/wikipedia/commons/5/54/Flight_dynamics_with_text.png
Tabulka 1	https://user-images.githubusercontent.com/37808065/38984279-37328b2c-4401-11e8-8e8e-30edcb94cccd.png
Tabulka 2	https://www.researchgate.net/publication/335043703/figure/tbl1/AS:789631830331395@1565274307666/You-Only-Look-One-v3-tiny-YOLOv3-tiny-network-structure.png

Odkazy na aplikace

Odkaz 1	https://github.com/pjreddie/darknet
Odkaz 2	https://github.com/AlexeyAB/darknet
Odkaz 3	https://github.com/penny4860/tf2-eager-yolo3
Odkaz 4	https://github.com/tzutalin/labelImg
Odkaz 5	https://github.com/54LiNKeR/boobs
Odkaz 6	https://github.com/Microsoft/VoTT#exporting-labels
Odkaz 7	https://gitlab.com/vgg/via
Odkaz 8	https://labelbox.com/
Odkaz 9	https://supervise.ly/
Odkaz 10	https://github.com/wkentaro/labelme

Adresářová struktura přiloženého disku

/dp	diplomová práce ve formátu .pdf
/aplikace	zdrojové kódy aplikace
/aplikace-YOLO	zdrojové kódy aplikace