VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE A ROZPOZNÁNÍ ZBRANĚ VE SCÉNĚ

DETECTION AND RECOGNITION OF GUN IN A SCENE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. DAVID STUCHLÍK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. MARTIN DRAHANSKÝ, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Stuchlík David, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Detekce a rozpoznání zbraně ve scéně**
Detection and Recognition of Gun in a Scene
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte literaturu týkající se detekce objektů zájmu ve scéně, primárně se zaměřte na zbraně.
2. Navrhněte algoritmus pro detekci a rozpoznání typu zbraně (krátká, dlouhá, automatická) v obrázku.
3. Navržený algoritmus z předchozího bodu implementujte.
4. Proveďte otestování na alespoň 500 obrázcích a vymezte se oproti existujícím řešením. Dosažené výsledky shrňte.

Literatura:

- OLMOS, Roberto; TABIK, Siham; HERRERA, Francisco. Automatic handgun detection alarm in videos using deep learning. *Neurocomputing*, 2018, 275: 66-72.
- SINGLETON, Mitchell, et al. Gun Identification Using Tensorflow. In: *International Conference on Machine Learning and Intelligent Communications*. Springer, Cham, 2018. p. 3-12.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cílem diplomové práce je návrh algoritmu pro detekci a rozpoznání typu zbraně v obrázku. V textu práce jsou nejprve stručně představeny existující metody a techniky detekce různých objektů, primárně jsou však metody zaměřeny na zbraně. Dále jsou stručně nastíněny základy neuronových sítí následované přehledem nejběžnějších detektorů pro hloubkové neuronové sítě. Druhá polovina práce se věnuje implementaci aplikace pro generování obrázků na základě 3D modelu zbraně, tvorbě datového souboru a učení neuronové sítě. V závěru práce jsou stručně shrnuty dosažené výsledky, které jasně indikují, že pro pokrytí obrovské variace reálných zbraní je zapotřebí vygenerovat velké množství trénovacích dat na základě mnoha různých 3D modelů.

Abstract

The aim of the diploma thesis is to design an algorithm for detection and recognition of the type of gun in the image. Firstly, the existing methods and techniques for detecting the various objects are briefly introduced in the text of the thesis however, the methods are primarily focused on guns. Next, the basics of neural networks are briefly outlined, followed by an overview of the most common detectors for deep neural networks. The second half of the thesis is devoted to the implementation of an application for generating images based on a 3D model of a gun, the creation of a data file and learning of a neural network. Finally, the results obtained, which clearly indicate that in order to cover a huge variation of real weapons, is necessary to generate a large amount of training data based on many different 3D models, are briefly summarized in the conclusion of the thesis.

Klíčová slova

detekce zbraní, počítačové vidění, strojové učení, zpracování obrazu, hluboké neuronové sítě, RetinaNet, TensorFlow, Three.js, Keras RetinaNet

Keywords

gun recognation, computer vision, machine learning, image processing, deep neural networrks, RetinaNet, TensorFlow, Three.js, Keras RetinaNet

Citace

STUCHLÍK, David. *Detekce a rozpoznání zbraně ve scéně*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Martin Drahanský, Ph.D.

Detekce a rozpoznání zbraně ve scéně

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Martina Drahanského Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Stuchlík
3. června 2020

Poděkování

Rád bych poděkoval vedoucímu práce panu prof. Ing. Martinou Drahanskému Ph.D., za oborné vedení a cenné rady při tvorbě diplomové práce.

Obsah

Kapitola 1

Úvod

Pojmy jako detekce objektů nebo strojové učení jsou v posledních několika letech často probíraná témata. S detekcí objektů se lze setkat v mnoha odvětvích každodenního života. Ať je to ovládání televizoru pomocí gest, odemčení chytrého telefonu pomocí obličeje nebo třeba automatického rozpoznání SPZ automobilu a umožnění vjezdu na parkoviště.

Jedním z velice zajímavých objektů pro detekci mohou být i zbraně. Několik psychologických studií prokázalo, že lidé, kteří mají přístup ke zbrani, mají několikanásobně větší pravděpodobnost spáchání násilného chování [60]. Systém, umožnující včasnou detekci útočníka se zbraní a následné upozornění na tuto skutečnost odpovídající bezpečnostní složky, může mít pozitivní dopad na počet obětí tohoto incidentu. Pokud by byl navíc tento systém schopen automaticky rozpoznat o jaký druh palné zbraně se jedná, může hrát systém významnou roli v rozhodování, jaký typ bezpečnostních složek k incidentu povolat.

Jednou z otázek, která každého napadne v souvislosti s detekcí objektů je: „Jaké jsou potíže a problémy v detekci objektů?“ Na tuto otázku však není úplně jednoduché odpovědět. Každý detekční úkol má zcela odlišné cíle a omezení, které se mohou navzájem lišit. Kromě obecných výzev spojených s vizuálními úkoly, jako rozdílné osvětlení, variace objektů spadajících do stejné třídy, nebo objekty pod různým úhlem pohledu, existují i specifické problémy související s konkrétním objektem. Další problém, který může nastat, kromě výše uvedených, je manipulace se zbraní jednou nebo oběma rukama, a tak může být značná část zbraně skryta.

Systém pro detekci zbraní by se mohl v blízké době stát běžnou součástí monitorovacích systému na veřejných prostranství a míst s větší koncentrací lidí, kde by mohlo docházet k ozbrojeným útokům. Monitorovací systém umístěný například na fotbalovém stadionu nebo na běžné poště či bance by mohl sám detekovat potencionálního útočníka a mobilizovat tak patřičné bezpečností složky.

Cílem této diplomové práce bylo prostudovat literaturu týkající se detekce objektů zájmů ve scéně, primárně se zaměřením na zbraně. Následně navrhnout a implementovat algoritmus, pro detekci a rozpoznání typu zbraně v obrázku. Posledním úkolem bylo otestování algoritmu alespoň na 500 obrázcích a porovnání výsledků proti existujícím řešením.

V úvodu kapitoly 2 je stručně popsán historický vývoj metod spadajících do oblasti detekce objektů v obraze. V podkapitole je popsáno několik existujících řešení detekce zbraní založených na rentgenových snímcích, segmentaci obrazu a s touto prací spjatou metodou hloubkového učení. Následuje podkapitola ve které jsou stručně vysvětleny pojmy jako umělá inteligence, strojové učené a neuronová síť a její základní stavební části. V úplném závěru této kapitoly je popsán princip hloubkového učení a do této kategorie spadajících konvoluční neuronová síť včetně popisu tří základních neuronových vrstev.

Kapitola 3 je věnována základním architekturám a detektorům založených na konvolučních neuronových sítích. V úvodu této kapitoly jsou stručně popsány jednotlivé soutěže a s tím spojené datové soubory na kterých jsou konkretní metody vzájemně porovnávány. Následuje stručný popis základních páteřních sítích na kterých jsou jednotlivé detektory postaveny. V druhé části této kapitoly jsou popsány jednotlivé detektory určené pro detekci objektů.

V kapitole 4 je popsán algoritmus pro generování snímků zbraní na základě 3D modelu zbraně. Následuje popis generování datového souboru a trénování modelu založeného na RetinaNet pomocí knihovny Keras RetinaNet. V závěru kapitoly je popsán algoritmus pro detekci a zbraní v obraze pomocí natrénovaného modelu.

Kapitola 5 je věnována testování natrénovaného modelu na různých datových vstupech. Následně je provedeno vyhodnocení natrénovaného modelu a porovnání výsledků s již existujícími řešeními.

V poslední kapitole 6 jsou shrnuty výsledky a zhodnocena celková práce. Dále jsou shrnuty nedostatky práce a možnosti budoucího rozšíření a vylepšení.

Kapitola 2

Metody pro detekci objektů v obraze

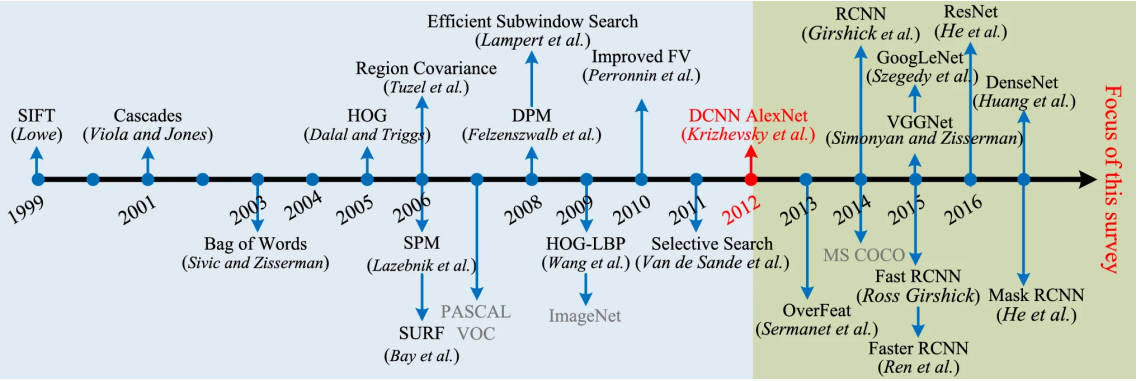
Detekce objektů je jedním ze zásadních a náročných problémů v oblasti počítačového vidění. V posledních několika letech se stala aktivní oblastí výzkumu [35][11]. Cílem počítačového vidění je zjistit, zda se ve zkoumané oblasti nachází požadovaný objekt spadající do dané kategorie zájmů. Pokud je takový objekt nalezen, je získáno jeho prostorové umístění a rozsah všech instancí nalezeného objektu např. prostřednictvím rámečku kolem detekovaného objektu. Detekce objektů tvoří základní kámen pro řešení složitých vizuálních úkolů, mezi které patří segmentace obrazu, sledování objektů, detekce událostí atd. Detekce objektů je také využívaná v celé řadě aplikací včetně robotického vidění, autonomního řízení, bezpečnostních systému a mnoho dalších.

Detekce objektů může být seskupena do dvou kategorií [67][69]. Detekce konkrétních objektů nebo detekce několika objektů různých kategorií. První typ si klade za cíl odhalit instance konkrétních objektů jako například obličej známe osobnosti nebo specifický model automobilu. Naproti tomu druhý typ si klade za cíl odhalit instance některých předdefinovaných kategorií objektů jako jsou lidi, zvířata atd. Tento typ se stává v posledních letech čím dál více populární a vznikají systémy, které jsou schopny detekovat širokou skupinu objektů pro všeobecné účely a soupeří tak s lidskou schopností detekce více objektů zároveň.

Z hlediska časového vývoje lze detekci objektů rozdělit na dvě historická období. Období před rokem 2012, kdy se pro pro detekci objektů využívaly tradiční metody detekce, a období po roce 2012, kdy začíná období detekce založené na hloubkovém učení [44][21][35][69]. Většina tradičních algoritmů byla postavena na základě ručně vytvořených funkcích. Kvůli nedostatku výpočetního výkonu a nedostatečné obrazové reprezentace nebylo možné využití jiných metod.

2.1 Vývoj metod pro detekci objektů

V této části bude velice stručně přestaven vývoj metod umožnující detekci objektů. Jelikož je toto téma velmi obsáhlé a existuje nespočet různých metod a přístupů budou zde některé metody pouze vyjmenovány s odkazem na patřičnou literaturu. Na obrázku 2.1 je znázorněna časová osa znázorňující představení jednotlivých metod detekce objektů.



Obrázek 2.1: Časová osa vývoje jednotlivých detekčních metod a algoritmů [35].

2.1.1 Detekce objektů pomocí statistických klasifikátorů, zájmových bodů, deskriptorů

Jednu skupinu tvoří metody využívajících ke své funkci statistické klasifikátory jako, jsou neuronové sítě [53], SVM [46] a Adaboost [13]. Tato úspěšná rodina detektorů objektů připravila půdu pro další výzkum v této oblasti.

Další skupinou jsou ručně vytvořené invariantní rysy. Ty získaly obrovskou popularitu například z důvodů invariance ke změnám měřítka, rotace a osvětlení. Počínaje metodou Scale Invariant Feature Transfrom (SIFT), kterou představili D. G. Lowe [38]. Zpřesnění této metody vedlo k tomu, že přístup SIFT je široce používán pro rozpoznávání objektů ve 2D obrazech. Pokrok v různých oblastech vizuálního rozpoznávání byl založen na použití zájmových deskriptorů, jako jsou rysy podobné Harrisovým [65][64], SIFT [6], histogram gradientů (HOG) [10] atd. Tyto lokální rysy jsou obvykle agregovány jednoduchým zřetězením nebo sdružováním, jako je přístup Bag of Visual Words [6].

Detekce objektů pomocí zájmových bodů a deskriptorů je dobře známý přístup. Schmid a Mohr [54] navrhli použití Harrisových rysů [22] jako bodů zájmu v šedotónovém obraze. Zájmové body byly poté charakterizovány pomocí řady deskriptorů, které byly poté uloženy v hašovací tabulce. Rozpoznání začíná generováním zájmových bodů a jejich deskriptorů. Následně dochází k vyhledávání ve vstupním obraze a následně jejich vyhledáním v hašovací tabulce.

Výše zmíněné metody dominovaly oblasti počítačového vidění mnoho let. Zásadní přelom přišel v roce 2012, kdy na soutěži ImageNet Large Scale Visual Recognition jednoznačně zvítězila metoda [29], kterou představil Alexandr Krizhevsky a jeho tým.

Viola-Jones detektor

Jednu z nejznámějších metod představili autoři článku P.Viola a M.Jones, kteří bez jakýchkoliv omezení detekovali lidskou tvář [65]. Detektor, který dostal název po svých autorech, byl desetinásobně, v některých případech i stonásobně rychlejší, než existující algoritmy.

Algoritmus využívá posuvného okna, které prochází celým obrázkem a rozhoduje, zda se v dané oblasti nachází lidská tvář. I když se může zdát, že se jedná o velmi jednoduchý proces, byla jeho výpočetní síla daleko za hranicemi své doby. Detekční rychlost tohoto algoritmu byla dramaticky zvýšena díky třem použitým technikám:

- Integrální obraz - výpočetní metoda sloužící k urychlení procesu filtrování nebo konvoluce.

- Výběrová funkce - místo použití sady ručně vybraných filtrů využili autoři algoritmus Adaboost[13], který vybere pouze malou sadu funkcí, které jsou užitečné pro detekci obličeje.
- Detekční kaskády - vícefázové detekční paradigma bylo zavedeno pro snížení výpočetní režie. Pokud v průběhu klasifikace jakákoliv úroveň klasifikuje danou oblast tak, že se v ní objekt nenachází, je výsledek detekce na přítomnost daného objektu vyhodnocen jako negativní.

Metody založené na hloubkovém učení

I když byly konvoluční neuronové sítě známy již dříve, jedním z hlavních impulzů pro jejich výzkum byl okamžik, kdy v roce 2012 Alexandr Krizhevsky a jeho tým [29], zvítězil v soutěži ImageNet Large Scale Visual Recognition (ILSVR) s rekordním náskokem. Jejich síť s názvem AlexNet byla složena z pěti konvolučních vrstev, tří pooling vrstev a tří plně propojených vrstev. Od té doby se výzkum počítačového vidění především zaměřuje na metody hlubokého učení [35].

Hluboké učení umožňuje výpočetním modelům učit se složité, jemné a abstraktní reprezentace, což vede k významnému pokroku v celé řadě problémů, jako jsou vizuální rozpoznávání, detekce objektů, rozpoznávání řeči, zpracování přirozeného jazyka, analýza lékařských obrazů, objevování léků a genetika. Mezi různými typy hlubokých neuronových sítí dosáhly CNN průlomy ve zpracování obrázků, videa, řeči a zvuku.

2.2 Existující metody detekce zbraní

Tato kapitola pojednává o existujících přístupech pro detekci zbraní. První a tradiční podoblast v detekci zbraní se zaměřuje na detekci skrytých zbraní v rentgenových obrazech. Druhou oblast tvoří metody založené na odstranění nežádoucích objektů v segmentovaném obrazu pomocí algoritmů shlukování a následně využívají některou z metod pro detekci objektu zájmů. Všechny výše uvedené systémy jsou pomalé, nelze je použít pro neustálé monitorování, vyžadují dohled obsluhy, a nelze je používat na otevřených prostranstvích. Poslední oblastí, která se stala za posledních několik let velmi populární a dosahuje skvělých výsledků, jsou metody založené na hloubkovém učení. Tyto metody nejčastěji využívají detektory založené na konvolučních neuronových sítích, které jsou schopné extrahovat mnoho vlastností. Dále budou stručně představeny vybrané metody z každé oblasti.

2.2.1 Rentgenové snímky

Detekce a klasifikace zbraní pomocí rentgenových technologií existuje již řadu let. Představeno bylo velké množství přístupů a technologií [16],[2][43][5]. Klasifikace obrazů na základě rentgenových snímků je v počítačovém vidění náročným úkolem, neboť snímky z rentgenového záření jsou běžně obklopeny jinými objekty a při otočení je nelze snadno rozpoznat [7]. Další metody popsané dále, které lze využívat v zařízeních pro odbavování zavazadel na letištích. Existuje několik metod dosahujících vysoké přesnosti. Jelikož je většina nalezených přístupů založena na detekci kovů, nelze detekovat nekovové zbraně. Pořizovací náklady jsou vysoké, neboť fungují pouze v kombinaci s rtg skenery a pásovými dopravníky.

Baştan a kol. [5] aplikovali model strojového učení typu bag-of-color na barevné 2D rentgenové snímky zavazadel pro detekci zbraní. Rentgenové skenery osvětlují zavazadlový předmět rentgenovým paprskem s vysokým a nízkým výkonem, z něhož lze odhadnout typ

materiálu přítomného v každém umístění pixelu. Výsledkem je obrázek který je obarven v závislosti na typu materiálu a pomáhá tak lidským dělníkům rozpoznávat objekty. Bylo provedeno zkoumání různých detektorů (DoG, Hessian-Laplace, Harris, FAST, STAR) spojených se třemi deskriptory (SIFT, SURF, BRIEF).

Proces, kterým jsou objekty v rentgenových snímcích barveny na základě jejich hustoty, se nazývá pseudobarvení [2]. Této techniky využívají i autoři článku [37]. Pseudobarevné rentgenové snímky jsou nejprve prochází procesem předzpracování. Poté jsou pomocí Analýzy propojených komponent získány komponenty, které obsahují kovové předměty a stávají se tak kandidátem na výskyt zbraní. Jednotlivé prvky byly dále extrahovány za pomoci Zernikových Momentů a deskriptoru kontextu. Získané prvky sloužily jako vstup do klasifikátoru ANFIS. Autoři metody sice udávají přesnost 90 %, ale trénování probíhalo pouze na 100 obrázcích zavazadel obsahující zbraň a 200 obrázcích zavazadel beze zbraně. Rovněž jsou uvedené obrázky zbraní zachyceny z boční strany a zachycují tedy maximální plochu zbraně bez překrytí.

Další metodu založenou na detekci ručních zbraní v rentgenových snímcích představili Nercessian a kol. [41]. Metoda používá detekci hran k lokalizaci zbraně. Avšak tato metoda se zabývá pouze ručními zbraněmi v pevně dané orientaci, dále byl použit malý dataset (40 obrázků s ručními zbraněmi, 400 obrázků neobsahující zbraň). Autoři článku nepředložili žádné statistické výsledky.

Poslední zmíněnou metodu v této oblasti publikovali autoři Oertel a Bock [43], kteří řeší detekci ruční zbraně ve 2D rentgenovém snímku zavazadel v šedé škále. Výzkum je příkladem rozpoznávání konkrétních položek: jeden typ pistole byl charakterizován pomocí rozlišovacích znaků jako je spoušť, zásobník nebo kohoutek. Zájmové oblasti jsou vytvořeny pro každý pixel z obrysu objektu. Stejně jako v předchozí metodě byl použit malý datový soubor (40 rentgenových snímků: 30 pro výcvik a 10 pro testování) a nebyly získány žádné kvantitativní výsledky.

2.2.2 Metody založené na segmentaci obrazu

Harrisův detektor

První představenou metodou pro detekci zbraní založené na segmentaci barev je metoda, publikovaná autory Rohit Kumar Tiwari a Gyanendra K. Verma [64]. Pro extrakci vlastností daného objektu byla použita kombinace Harrisova detektoru společně s FREAK extraktorem vlastností. Harrisův detektor byl použit, protože je invariantní ke geometrické transformaci a částečně rezistentní ke změně osvětlení a šumu, zatímco FREAK popisuje klíčové body. Je inspirován lidským vizuálním systémem a je používán pro výpočet kaskády binárních řetězců efektivním porovnáním intenzit obrazu. Hlavní výhodou představené metody je rychlost.[3]

Ze vstupního obrazu je nejprve odstraněn šum pomocí mediánového filtru, poté je provedena změna velikosti na 400 x 300 pixelů. Pomocí shlukovacího agoritmu k-means je provedena segmentace založená na barvě. Tato segmentace se provádí za účelem získání barvy související se zbraní. Díky tomu dochází k eliminaci objektů podobajících se zbrani. Následuje extrakce objektů ze segmentovaného obrazu. Z důvodu odstranění nežádoucího šumu jsou extrahovány pouze objekty, které mají větší plochu než 1000 pixelů. Pro odstranění malých mezer v extrahovaném objektu je provedeno morfologické uzavření. Z takto připraveného bloku jsou extrahovány hranice, které budou použity při porovnávání. Porovnávání objektu pomocí hranic je výhodné z toho důvodu, že vnitřní struktura objektu se může lišit, ale obrys objektu zůstává stejný nebo se liší jen nepatrně. Poslední fází zmí-

něné metody je porovnávání mezi uloženým deskritotem zbraně a extrahovaným objektem. Pro výpočet shody mezi dvěma deskriptory se využívá součet čtvercových rozdílů (SSD). Pokud je shoda mezi deskriptorem a extrahovaným objektem alespoň 50 % je objekt vyhodnocen jako zbraň.

Výhodou této metody je detekce více zbraní v jednom obraze a díky nastavené shodě na hodnotu 50 % lze detekovat i částečně zakrytou zbraň. Avšak daná metoda neumožňuje detekci zbraní různých tvarů či atypických barev. Dále nedokáže rozpoznat odlišně orientované zbraně než je uložený descriptor.

Sledovací systém Bag of Words

Další metodu pro detekci zbraní využívající k-means algoritmus představili Nadhir Ben Halima a kol. [6]. Tito autoři používají zcela odlišný algoritmus pro extrakci hlavních oblastí obrazu - algoritmus SIFT. Extrakční algoritmus SIFT detekuje významné oblasti obrazu nazývané klíčové body a provádí se ve čtyřech fázích:

- detekce extrémního měřítka,
- lokalizace klíčových bodů,
- orientační přiřazení,
- deskriptor klíčových bodů.

Extrahované descriptory pomocí algoritmu SIFT jsou odolné vůči rozdílnému osvětlení a invariantní vůči změně zobrazovaných bodů, jako jsou například rotace nebo změna měřítka.

Navrhovaný algoritmus nejprve ze sady trénovacích obrazů extrahuje vektory lokálních prvků SIFT. Následně ze všech extrahovaných vektorů vytvoří jednu sadu. Takto vytvořená sada tvoří vstup shlukovacímu algoritmu k-means, pomocí kterého se získá sada klastrů, kde každý klastr reprezentuje jedno vizuální slovo. Dalším krokem je výpočet prostorového histogramu, který udává, kolikrát se jedno vizuální slovo vyskytuje v každém obrázku. Funkce prostorového histogramu z každého obrázku jsou následně předána jako vstup k tréningu SVM klasifikátoru, který nové obrázky klasifikuje. Nakonec je detekována pozice zbraně v obrázku. Testovací data ani přesnost algoritmu není v představené metodě jasně specifikována.

Klasifikace pomocí algoritmu detekce hran a zpracování obrazu na nízké úrovni SUSAN

Tato metoda detekce, kterou publikovali Wang a kol. [28], pracuje na principu porovnávání získaných obrazových bloků se skupinou existujících obrazových modelů.

Jelikož je popsaná metoda navržena pro zpracování obrazů ve formátu JPG a rozlišení 250x250 pixelů musí nejprve všechny obrazy projít procesem předzpracování z důvodu rychlejší detekce. Nejprve dojde ke zmenšení vstupního obrazu na rozlišení 250x250 pixelů a následně jeho převedení do odstínů šedi, poté je obraz zpracován Cannyho hranovým detektorem [39]. Pomocí tohoto detektoru jsou získány okraje objektu o tloušťce 1 pixelu. V takto předzpracovaném obrazu dochází k následné identifikaci hran objektu pomocí Susan detektoru [59]. Výsledky tohoto zpracování porovnává s databází již existujících modelů. Nalezení odpovídajících částí je vykonáváno za pomoci algoritmů přizpůsobení bloku a úplného vyhledávání. Následně provede statistický výpočet, na jehož výstupu nalezneme klasifikaci zbraně. Výsledný model je přidán do databáze a bude využit pro budoucí porovnávání.

Autoři článku udávají klasifikaci poloautomatické zbraně s přesností okolo 99 % a klasifikaci zbraně okolo 81 %. Uvedených hodnot přesnosti klasifikace zbraní lze dosáhnout pouze za předpokladů, že je zbraň snímána ze strany a obraz zbraně je bez jakýchkoliv známek poškození

2.2.3 Automatická detekce zbraní pomocí Deep Learning

Detekce zbraní na architektuře VGG-16

Metodu založenou na architektuře VGG-16 publikoval Roberto Olmos a kol. [44], využívající k detekci objektů ve video záznamu metody hloubkového učení. Hlavní cíl metody spočívá v nalezení nejvhodnějšího detektoru založeného na CNN, který umožní detekci zbraně v reálném čase.

Toto kritérium nejlépe splňoval FasterR-CNN detektor který je založen na přístupu regionálních návrhů. Detektor kombinuje metodu selektivního vyhledávání s klasifikátorem založeným na architektuře VGG-16. Klasifikátor byl přetrénovaný na databázi ImageNet, která obsahuje přibližně 15 milionů obrázků zařazených do přibližně 22 tisíc tříd. Následně proběhlo trénování klasifikátoru na zbraně. K tomuto účelu byla vytvořena databáze obsahující 3000 obrázků zbraní v různém prostředí.

Představená metoda byla testována na 7 videích horší kvality. Metoda vykazuje přesnost přibližně 85 %. Eliminace falešně pozitivních poplachů je částečné eliminován tím, že dochází k upozornění na vyskytující se zbraň až ve chvíli, kdy je v 5 po sobě jdoucích snímcích zbraň nalezena.

Detekce zbraní pomocí Tensorflow

Metodu automatické detekce zbraní pomocí platformy Tensorflow představili Mitchell Singleton a kol. [58] ve svém článku. Tato platforma je vytvořena společností Google a je využívaná v celé řadě projektů.

Pro detekci objektů zde využívají model MobileNet. Jedná se v podstatě o kolekci modelů, které jsou orientovaný pro mobilní zařízení. Na rozdíl od klasické CNN rozděluje konvoluci na hloubkovou konvoluci 3 x 3 a bodovou konvoluci 1 x 1. Díky tomu se výpočetní čas ve srovnání s tradiční CNN významně zkrátí, ale jeho přesnost se sníží. Záleží však na použité architektuře [26].

Pro účely trénování bylo nashromážděno přibližně 3400 obrázků zbraní a necelých 12 000 obrázků z různých zdrojů, které neobsahují zbraň. Trénování probíhalo na několika odlišných modelech: Inceptionv3, MobileNet0.50.192, MobileNet1.0.224 atd. Po dokončení trénovací fáze byla použita metoda Shi Tomasi k identifikaci klíčových bodů, které budou zkoumány na přítomnost zbraně. Každý klíčový bod se následně používá k oříznutí části obrazu, ve které se zkoumá, zda obsahuje zbraň či nikoliv. Pokud algoritmus nalezne v dané části zbraň, dojde k barevnému ohraničení této části.

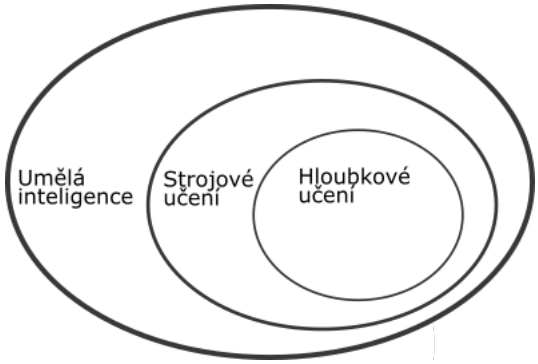
Model byl trénován různým počtem iterací za účelem získání co možná nejlepší přesnosti. Z výsledků je patrné, že pokud byl model trénován pomocí více iterací, dojde ke zpřesnění. Při počtu 3000 iterací byla přesnost 97.89 %, kdežto při počtu iterací 5000 dosahovala přesnost 100 %. Testování ukázalo, že přesnost metody na obrazcích neobsahující zbraň je přibližně 96 % a v případě obrázků obsahující zbraň kolem 89 %.

2.3 Umělá inteligence, strojové učení a hloubkové učení

V této části budou nejprve ve stručnosti vysvětleny základní pojmy, jako je umělá inteligence, strojové učení a hloubkové učení. Dále budou představeny Konvoluční neuronové sítě.

2.3.1 Umělá inteligence

Umělá inteligence má za snahu automatizovat intelektuální úkoly, které lidé běžně vykonávají. Umělou inteligenci můžeme chápat jako obecný obor, který v sobě zahrnuje strojové učení, hloubkové učení a mnoho dalších principů 2.2 . Dost dlouhou se myslelo, že umělá inteligence na úrovni lidského myšlení může být dosažena dostatečně velkou sadou explicitních pravidel. Tento přístup je známý jako symbolická umělá inteligence. V průběhu času se však ukázalo, že vymyslet explicitní pravidla pro klasifikaci obrazu nebo rozpoznávání řeči je značně složité, proto se objevil nový způsob, který zaujal místo symbolické umělé inteligence a tím je strojové učení [8].



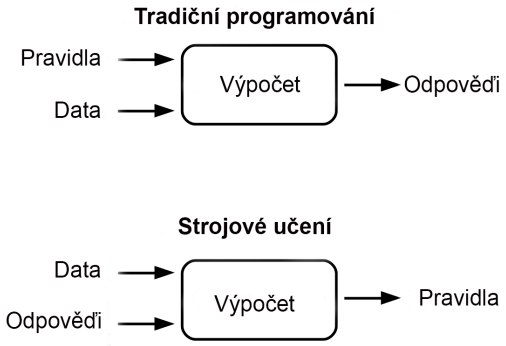
Obrázek 2.2: Obrázek znázorňuje vzájemný vztah mezi umělou inteligencí, strojovým učením a hloubkovým učením.

2.3.2 Strojové učení

Strojové učení je tedy podoblastí umělé inteligence zabývající se algoritmy a technikami, které na základě vstupních dat a očekávaných odpovědí vytvoří pravidla. Tato pravidla mohou být následně použita na nová data a vytvářet tak originální odpovědi [8]. Tento princip je zcela odlišný od klasického programování. Při klasickém programování vkládáme data a pravidla a výstupem programu je nějaká odpověď. Při strojovém učení je situace odlišná. Vstupem algoritmu je datová sada a sada odpovědí. Algoritmus se sám snaží nalézt pravidla.

Aby bylo možné tyto pravidla nalézt vyžaduje metoda strojového učení tři podmínky:

- Vstupní data - pokud je například úkolem rozpoznání objektu v obraze je zapotřebí dodat jako vstupní data soubor obrázků, na kterých se bude učit.
- Soubor očekávaných dat - mohou to být například značky s druhy zvířat (kočka, pes, prase atd.).
- Způsob jak měřit, že algoritmus funguje - je nutné určit, zda výstup algoritmu odpovídá očekávanému výstupu.



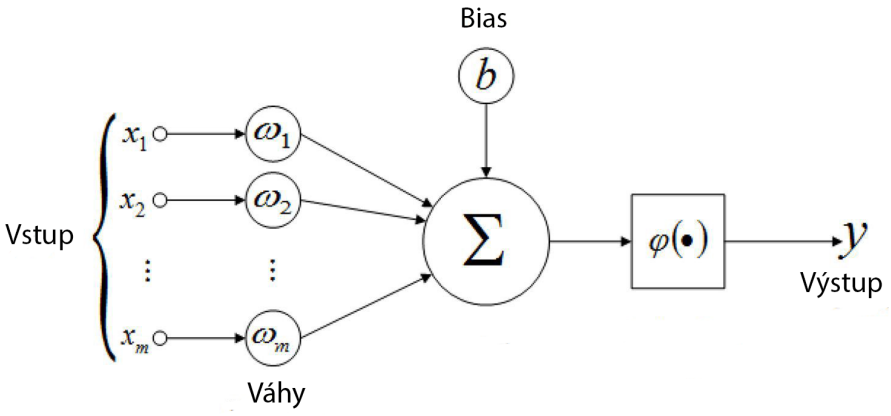
Obrázek 2.3: Znázornění rozdílu mezi klasickým programováním a metodou strojového učení

2.3.3 Neuronová síť

Umělé neuronové sítě jsou biologicky inspirované výpočetní struktury spadající do oblasti umělé inteligence. Jejich hlavním cílem je shromažďovaní znalostí tím, že zjišťují vzorce a vztahy ve vstupních datech. Neuronové sítě se učí prostřednictvím zkušeností nikoli programováním. Umělé neuronové sítě jsou tvořeny umělými neurony.

Umělý neuron

Jedná se o matematickou funkci, která byla inspirovaná modelem biologického neuronu. Každý neuron má několik vstupů, které je možné přirovnat k dendritům neuronu lidské nervové soustavy. Přicházející signály jsou nejprve násobeny jednotlivými váhami a poté provedena jejich suma. Tato hodnota se označuje jako celková aktivace uzlu. Sčítaná aktivace je poté transformována pomocí aktivační funkce a definuje specifický výstup uzlu. Výstup aktivační funkce slouží jako vstup pro další vrstvu neuronů nebo pro vyhodnocení výsledku. Základní stavební bloky umělého neuronu jsou znázorněny na obrázku 2.4.



Obrázek 2.4: Znázornění rozdílu mezi klasickým programováním a metodou strojového učení [8].

Aktivační funkce

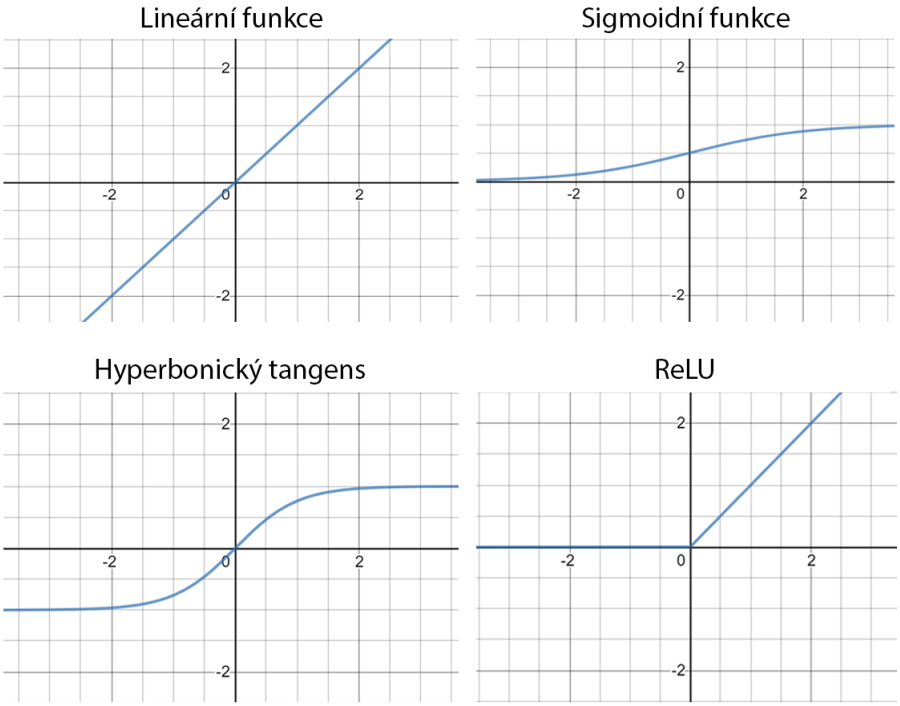
Nejjednodušší aktivační funkcí je lineární funkce, která nepoužívá žádnou transformaci. Síť složená z pouze z lineárních funkcí se snadno učí, ale vykazuje nedostatečnou schopnost učení složitých mapovacích funkcí. Lineární aktivační funkce jsou využívány ve výstupní vrstvě sítí zabývajících se problémy s regresí.

Pro učení složitějších datových struktur je vhodné využít některou z nelineárních aktivačních funkcí. Mezi dvě tradičně využívané aktivační funkce patří funkce Sigmoidní a Hyperbonický tangens.

Nelineární Sigmoidní funkce byla tradiční aktivační funkcí používanou v neuronových sítích. Vstupní hodnota je touto funkcí transformována na hodnotu mezi 0 a 1. Pokud je hodnota větší než 1 je tato hodnota transformována na hodnotu 1. Podobně je tomu s hodnotou menší něž 0, která je transformována na hodnotu 0.

Aktivační funkce Hyperbonický tangens nebo zkráceně tanh má podobné vlastnosti jako Sigmoidní funkce s tím rozdílem, že výstupní hodnota může nabývat hodnoty mezi -1 a 1. Tato funkce bývá často upřednostňována před Sigmoidní funkcí z důvodů snazšího učení a lepšího výkonu. Zásadní problém výše zmíněných nelineárních funkcí je takzvaný vanishing gradient [25].

Další aktivační funkcí, která vypadá a působí jako lineární funkce, ale ve skutečnosti je nelineární funkcí, umožňující se učit složité vztahy v datech, je funkce ReLU (Rectified linear unit). Tato funkce transformuje vstupní hodnotu na 0 pokud se jedná o zápornou hodnotu, v případě kladné hodnoty se transformace neprovádí. Výhodou této aktivační funkce je její snadná implementace, částečné lineární chování, rychlejší učení v porovnání s funkcemi Sigmoid nebo Hyperbonický tangens. Na obrázku 2.5 jsou znázorněny průběhy popsaných funkcí.



Obrázek 2.5: Grafické znázornění průběhů aktivačních funkcí.

Propojení neuronů

Vzájemné propojení neuronů má významný dopad na provoz umělé neuronové sítě. Propojení neuronů v síti může být zpravidla dopředné, kdy výstupy předchozí vrstvy ovlivňují přímo vstupy vrstvy následující, a proto neuchovává záznam o svých předchozích hodnotách. Druhou možností je zpětnovazební síť, která má spojení od výstupu zpět na vstupní neuron předchozí nebo vlastní vrstvy. Díky tomuto zpětnovazebnímu propojení má síť další možnost minimalizace chyby. Taková síť si udržuje hodnotu předchozího stavu, takže další stav závisí nejen na vstupní hodnotě, ale také na předchozím stavu sítě.

Princip učení

Existuje celé řada rozdílných pravidel pro učení, ale nejvíce využívané jsou pravidlo Delta nebo pravidlo Back-propagation. Neuronová síť je trénována k mapování sady vstupních dat iteračním nastavením vah. Optimalizace jednotlivých vah se provádí zpětným šířením chyby během tréninkové nebo učební fáze. Síť čte vstupní a výstupní hodnoty a mění hodnotu vážených odkazů, aby se snížil rozdíl mezi předpovězenými a cílovými hodnotami. Chyba v predikci je minimalizována během několika trénovacích cyklů, dokud síť nedosáhne stanovené úrovně přesnosti [48]. Pokud je však síť trénována příliš dlouho může dojít k přeučení a ztráty schopnosti zobecnění. Principů učení neuronových sítí je hned několik [4]. Jedním z nejvyužívanějších způsobů je učení s učitelem. V tomto případě jsou společně s vkládanými daty vkládána i požadovaná řešení, nazývaná štítky. Typickým příkladem takového typu učení je klasifikace. Opačným příkladem je učení bez učitele. V tomto případě nejsou vkládána požadovaná řešení, ale systém se pokouší sám vyhledávat a rozpoznávat souvislosti ve vstupních datech.

2.3.4 Hloubkové učení

Hloubkové učení je oblast strojového učení, která se pokouší učit na úrovni vysoké abstrakce s využitím hierarchické architektury. Hierarchická architektura může obsahovat desítky dokonce i stovky po sobě jdoucích vrstev. Tento přístup se využívá v základních oblastech umělé inteligence (počítačové vidění, sémantická analýza a mnoho dalších). Vzestup této oblasti je způsoben dramatickým zvýšením schopnosti zpracování dat (GPU jednotky), snížení ceny hardwaru a v neposlední řadě značný pokrok v algoritmech strojového učení [21][8]. V hloubkovém učení jsou tyto hierarchické architektury učeny prostřednictvím modelů nazvaných neuronové sítě.

V posledních letech je oblast hloubkového učení rozsáhle studována pro účely počítačového vidění. Díky tomu vzniklo několik různých přístupů. Obecně lze tyto přístupy rozdělit do několika kategorií. Mezi nejzákladnější patří:

- konvoluční neuronové sítě,
- omezený Boltzmannův stroj,
- autokodér.

2.3.5 Konvoluční neuronová síť

Konvoluční neuronová síť patří mezi nejvýznamnější přístup hloubkového učení. Tato síť se skládá z několika vrstev neuronů. Běžně nachází své využití v aplikacích počítačového vidění a je považován za velmi efektivní.

CNN architektura obsahuje tři hlavní nervové vrstvy:

- Konvoluční vrstvy,
- sdružující vrstvy,
- plně propojené vrstvy.

Každá z těchto vrstev plní různou funkci. Trénování probíhá ve dvou odlišných fázích. První fáze (fáze dopředná) - hlavním účelem je reprezentace vstupního obrazu s danými parametry v každé vrstvě. Následně je předpovídaný výstup použit pro výpočet ztrátové funkce. Na základě této hodnoty jsou ve zpětné fázi vypočítány gradienty všech parametrů. Nakonec jsou všechny parametry upraveny na základě vypočtených gradientů a připraveny pro další dopřednou fázi. Po několika úspěšných iteracích dopředné a zpětné fáze může být učení zastaveno.

Konvoluční vrstva

Cílem konvoluční vrstvy je získat ze vstupního obrazu vlastnosti na vysoké úrovni. Mezi tyto vlastnosti řadíme hrany, orientace přechodu, barvy atd. Konvoluční vrstva postupně prochází celý obraz a aplikuje na něho zvolené jádro. Jedná se v podstatně o násobení odpovídajících pixelů obrazu s hodnotami jádra. Jednotlivé výsledky násobení se poté sečtou a vznikne výsledná hodnota. Následuje posun jádra a dochází k opakování celého procesu dokud není operace aplikována na celý obraz. Konvoluční neuronové sítě používají i několik různých jader pro generování různých map vlastností. Princip funkce konvoluční vrstvy je znázorněn na obrázku 2.6.

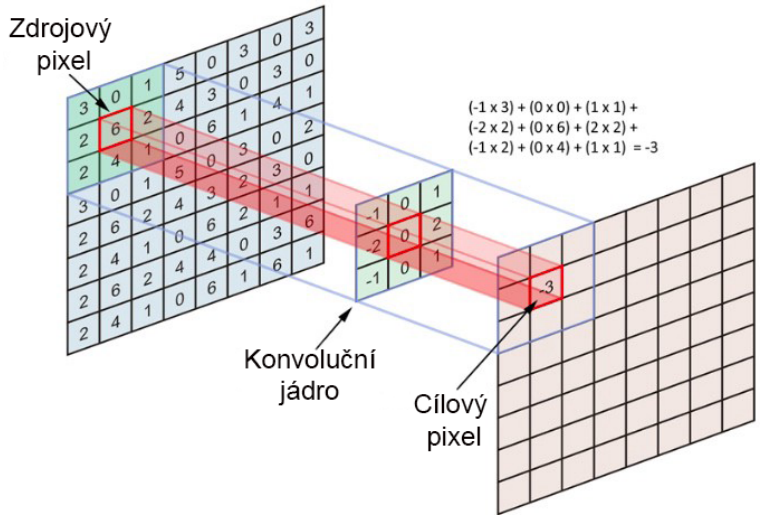
Konvoluční operace má tři hlavní výhody:

- mechanismus sdílení vah ve stejné mapě vlastností redukuje počet parametrů,
- lokální propojení se učí vzájemnou vazbu mezi sousedním pixelem,
- invariance k umístění objektu.

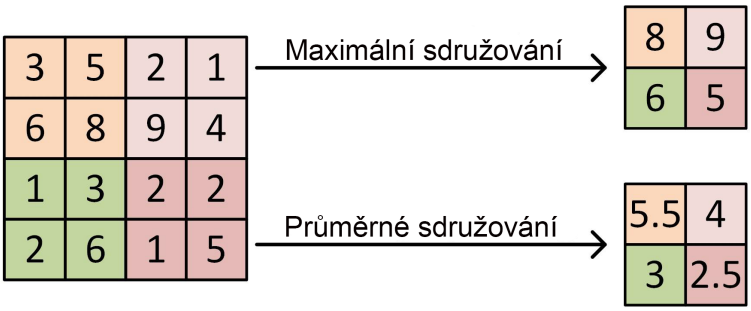
Sdružující vrstvy

Obecně je konvoluční vrstva následována sdružující vrstvou, pomocí které lze snížit počet dimenzí hlavních map a parametrů sítě. Díky tomu dochází ke snížení výpočetního výkonu potřebného pro zpracování dat. Podobně jako konvoluční vrstva je sdružující vrstva rotačně a pozičně invariantní, jelikož jejich výpočet zohledňuje sousední pixely. Průměrné sdružovaní a maximální sdružování jsou dvě nejčastěji využívané strategie.

Jak již název napovídá maximální sdružování vrací maximální hodnotu z části obrazu překrývaného jádrem. Maximální sdružování se využívá k potlačení šumu společně s redukcí rozměrů. Na druhou stranu průměrné sdružování vrací průměr všech hodnot z části obrazu pokrytého jádrem, jak znázorňuje obrázek 2.7. Maximální sdružování má lepší výkon než průměrné sdružování. Konvoluční vrstva a sdružovací vrstva společně tvoří i-tou vrstvu konvoluční neuronové sítě. V závislosti na složitosti obrázků může být počet takových vrstev zvýšen pro další zachycení detailů na nízké úrovni, avšak za cenu větší výpočetní náročnosti.



Obrázek 2.6: Obrázek znázorňuje funkci konvoluční vrstvy, kdy je pomocí konvolučního jádra a vstupního obrazu vypočítána nová hodnota pixelu [47].



Obrázek 2.7: Obrázek znázorňuje rozdíl mezi funkcí maximálního sdružováním a funkcí průměrného sdružování [12].

Plně propojené vrstvy

Tato vrstva vezme výstup z předchozí vrstvy (sdružující vrstvy), například matici vlastností, a převede jej na vektor vlastností, který může být vstupem pro další fázi. Plně propojené vrstvy fungují jako tradiční neuronová síť a obsahují asi 90 % parametrů v CNN. Výstup této vrstvy může být využit například k získání pravděpodobnosti, zda určitý znak patří do dané kategorie, nebo ho využít jako vektor funkce pro následné zpracování.

Kapitola 3

Základní architektury a detektory konvolučních neuronových sítí

Základem každé neuronové sítě je soustava několika vzájemně navazujících a propojených vrstev. V průběhu několika let vznikala celá řada různých architektur. Aby bylo možné tyto sítě trénovat, bylo potřeba vytvořit velké datové soubory, na kterých se budou neuronové sítě učit. Tvorba velkých datových souborů je kritická pro vývoj pokročilých algoritmů počítačového vidění. Použití velkých datových sad hraje významnou roli v mnoha oblastech výzkumu, protože pomocí nich je možné srovnávat různé algoritmy a stanovit tak cíle pro řešení. V oblasti detekce objektů vzniklo v posledních deseti letech několik dobře známých datových souborů. Mezi ty nejznámější patří Pascal VOC, ImageNet Large Scale Visual Recognition Challenge, MS-COCO Detection Challenge atd.

Pascal Visual Object Classes Challenges (Pascal VOC) [14] byla jednou z nejvýznamnějších soutěží rané komunity počítačového vidění. Soutěž byla složena z několika úkolů zahrnujících klasifikaci obrazu, detekci objektů, sémantickou segmentaci a detekci akce. Datové sady PASCAL VOC obsahují 20 kategorií (osoba, pták, lahev atd.) rozložených do 11 000 obrázků. Těchto 20 kategorií lze považovat za zástupce 4 hlavní odvětví - vozidla, zvířata, předměty pro domácnost a lidé. V datovém souboru PASCAL VOC je anotováno přes 27 000 instancí objektů. V datovém souboru VOC2007 jsou jednotlivé třídy nevyvážené, například třída osoba je téměř 20krát vetší než nejmenší třída ovoce.

Následující datovou sadu představuje **Microsoft Common Objects in Context (MS COCO)** [34], který obsahuje obrázky vhodné pro detekci a segmentaci objektů vyskytujících se v každodenním životě. Datový soubor obsahuje 91 společných kategorií objektů, z nichž 82 má více než 5 000 označených instancí objektů. Tyto kategorie zahrnují 20 podkategorií vyskytujících se v datovém souboru PASCAL VOC. Celkem má datový soubor 2 500 000 instancí označených v 328 000 obrázcích. Datový soubor MS COCO také věnuje pozornost různým hlediskům, a všechny jeho objekty jsou v přirozeném prostředí, které poskytuje bohaté kontextové informace. Na rozdíl od populárního datového souboru ImageNet [29] má COCO méně kategorií, ale více případů na kategorii.

Náročné datové sady mohou pomoci k rychlejšímu pokroku s vizuálními úkoly a praktickými aplikacemi. Takovou to náročnou datovou sadou je i datový soubor ImageNet. Na tomto datovém souboru byla od roku 2010 do roku 2017 pořádána soutěž **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)** [29]. Úlohou ILSVRC bylo vyhodnotit schopnost algoritmu lokalizovat všechny instance všech cílových objektů pří-

tomných v obraze a správně je přiřadit. Datová sada pro ILSVRC2014 má 200 tříd a téměř 450 000 výcvikových obrazů, 20 000 ověřovacích obrazů a 40 000 testovacích obrazů.

Dalším datovým souborem, který vznikl v roce 2018 je **VisDrone2018** [68]. Tento datový soubor se skládá z obrázků a videí pořízených drony. Cílem tohoto datového souboru je postupovat v úlohách vizuálního porozumění na platformě dronů. Obrazy a video sekvence byly pořízeny v různých městech a v příměstských oblastech 14 různých měst po celé Číně. VisDrone2018 má velké množství malých předmětů, jako jsou auta, chodci a jízdní kola, což způsobí obtížnou detekci určitých kategorií. Konkrétně VisDrone2018 obsahuje 263 videoklipů a 10 209 obrázků, které se nepřekrývají s video sekvencemi.

Posledním zmíněným datovým souborem je **Open Images** [30], který obsahuje 9,2 milionu obrázků. Open Image verze 5 obsahuje celkem 16 milionů ohraničeních objektů pro 600 různých tříd, což z něho činí největší existující datový soubor s anotacemi. Velká část tohoto datového souboru byla ručně anotována profesionálními anotátory, z důvodů zajištění přesnosti a konzistence. Obrazy v tomto datovém souboru jsou velmi rozmanité a většinou obsahují složité scény s několika objekty. Tento datový soubor nabízí vizuální anotace vztahů, které ukazují dvojice objektů ve zvláštních vztazích (např. pivo na stole, žena hrající na kytaru atd.). Celkem má datový soubor 329 vztahových trojic s 391 073 vzorky.

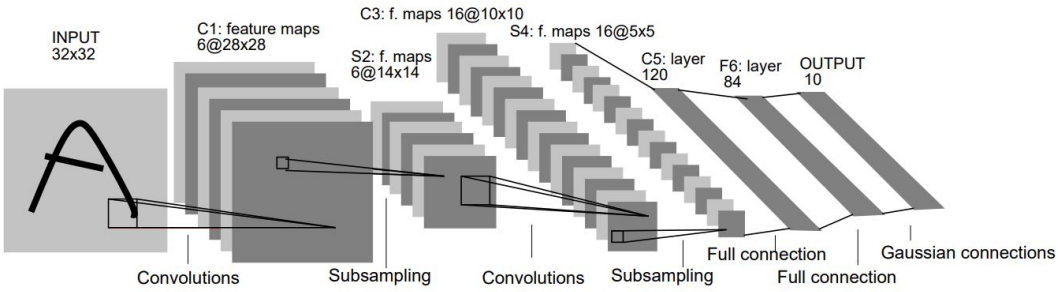
3.1 Páteřní sítě

Páteřní síť funguje jako základní extraktor vlastností pro detekci objektů, kde vstupním parametrem sítě jsou jednotlivé obrázky, a výstupem jsou mapy vlastností odpovídající vstupnímu obrazu. Většina páteřních sítí pro detekci objektů jsou sítě pro klasifikační úlohu vyřazující poslední plně propojené vrstvy.

3.1.1 LeNet-5

Ačkoliv zásadní průlom v hloubkovém učení pomocí neuronových sítí nastává v roce 2012, kdy v soutěži ILSVRC zvítězily autoři s architekturou AlexNet [29] vznikají úspěšné architektury již mnohem dříve. Například v roce 1998 Y.Lecun a kol. [31] vytvořili architekturu sítě pojmenovanou LeNet-5. Jelikož v minulosti nebyly k dispozici výkonné CPU nebo GPU, je tato architektura z dnešního pohledu velmi výpočetně nenáročná. Architektura LeNet se stala velmi úspěšnou a široce využívanou sítí pro rozpoznávání ručně psaných a tištěných číslic. Jak je znázorněno na obrázku 3.1 architektura je tvořena celkem 7 vrstvami. První a druhá konvoluční vrstva využívá 5 x 5 konvolučního jádra s posuvným krokem 1, třetí konvoluční vrstva využívá 1 x 1 konvoluční mapu s posuvným krokem 1. Sdružovací vrstvy využívají okno o velikosti 2 x 2 s krokem 2. Výstupní vrstva klasifikuje obraz pomocí softmax algoritmu do jedné z deseti tříd. Nelinearitu do sítě zavádí pomocí funkcí hyperbonického tangens nebo sigmoidů.

LeNet-5 byla jednou z úspěšných architektur na počátku vývoje CNN a představuje jeden z velkých kroků pro umělou inteligenci celkově. Tato architektura se stala základním kamenem pro řadu dalších sítí, mezi které patří i architektura AlexNet.



Obrázek 3.1: Architektura LeNet-5 je jedna z prvních architektur spadající do kategorie CNN, která sloužila pro rozpoznávání číslic [31].

3.1.2 AlexNet

Tato architektura se zapsala do historie CNN a je považována za jeden ze zásadních milníku v hloubkovém učení. V roce 2012 Alex Krizhevsky představil obdobu rozšířené architerktury LeNet nazvanou AlexNet [29]. Tato architektura ve stejném roce zvítězila v soutěži ILSVRC s chybovostí 15.3 %, čímž skončila se značným náskokem oproti architektuře umístěném na druhém místě s chybovostí 26.2 %. Krizkevsky využívá značného pokroku ve výpočetním výkonu, který mu přináší možnost trénování na větším množství dat. Toto dobré načasování umožnilo využití zmíněné architektury získávající asi šedesát milionů parametrů, ReLu vrstvy, maximálního sdružování a techniky droupout. Jako jedna z prvních sítí byla architektura AlexNet vyškolena na datovém souboru ImageNet. Síť je složena z pěti posupně propojených konvolučních vrstev a 3 plně propojených vrstev. Síť AlexNet bere vstupní obraz o velikost 227 x 227 x 3. Jednou z klíčových vlastností této sítě je rychlé převzorkovaní středních reprezentací.

ZF Net

Po úspěchu sítě AlexNet se konvoluční neuronové sítě staly velice zajímavým oborem. Matthew Zeiler a Rob Fergus vytvořili architekturu ZF Net [66], která byla založena na AlexNet a zvítězila na ILSVRC v roce 2013 s chybovostí 12.2 %. Autoři sítě zjistili, že při filtrování na první vrstvě AlexNet bylo ztraceno příliš mnoho informací, a poroto byla první konvoluční vrstva s konvolučním jádrem 11 x 11 nahrazena konvolučním jádrem o velikosti 7 x 7 s menším krokem. Došlo nejenom k vylepšení parametrů, ale také k rozšíření velikosti středních konvolučních vrstev. Autoři také použili ztrátovou funkci *cross-entrophy*. Důležitým rysem bylo zavedení technik pro mapování extrahovaných vlastností zpět na pixely. Tato technika se nazývá *deconvolutional network* a dává uživateli představu o tom, jaké struktury jsou rozpoznávány v jednotlivých mapách vlastností.

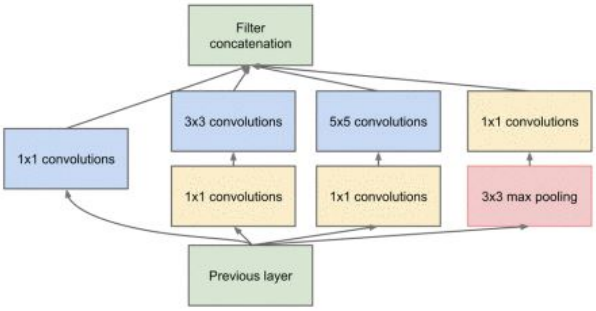
3.1.3 VGGNet

VGGNet představená roku 2014, navržená Visual geometry group of the University of Oxford, dosáhla míry chybovosti pouze 7.3 % [57]. Autoři architektury experimentovali s počtem vrstev v rozmezí 11 až 19. Výsledná architektura obsahovala 16 vrstev, kde konvoluční vrstvy využívaly konvoluční jádra o velikosti pouze 3 x 3. Rovněž byly vstupní obrazy doplňovány nulovými hodnotami pro zachování prostorového rozlišení po konvoluci prokládané vrstvami maximálního sdružování. Autoři architektury zjistili zajímavý fakt, a tím

je, že použití více konvolučních vrstev s menšími jádry v řadě, má podobný účinek jako použití větších jader se zachováním výhod menších jader. Například použití tří vrstev s jádrem 3 x 3 má podobný účinek jako využití jedné vrstvy s jádrem 7 x 7, avšak použitím 3 vrstev se sníží počet parametrů a umožní uživateli využít 3 vrstev ReLu místo původní jedné vrstvy.

3.1.4 GoogLeNet

Vítězem soutěže ILSVRC 2014 se s chybovostí 6.7 % stala architektura GoogLeNet [61]. Tato architektura využívá nového konceptu skládání sítě. Autoři zjistili, že sekvenční řazení sítě značně zvyšuje její výpočetní a paměťové nároky, a navrhli tak nový modul nazývaný *Inception*. Jak je znázorněno na obrázku 3.2, autoři vyřešili problém s hloubkou použitím konvolučních vrstev 1 x 1 označovaných také jako *síť v sítí*. Modul Inception využívá filtrů o velikostech 1 x 1, 3 x 3 a 5 x 5, toto rozhodnutí bylo založeno na snazší práci, nikoli nezbytnosti. Navrhovaná architektura je kombinací všech těchto vrstev, s jejichž výstupy zřetězenými do jediného výstupního vektoru tvoří vstup další fáze. Tato paralelní architektura používá více než 100 vrstev a její hloubka představuje pouze zlomek tohoto počtu vrstev. Architektura využívá jen 7 Inception modulů. Autoři také odstranili přebytečné plně propojené vrstvy a nahradili je vrstvami průměrného sdružování, čímž razantně snížili počet parametrů oproti síti AlexNet.



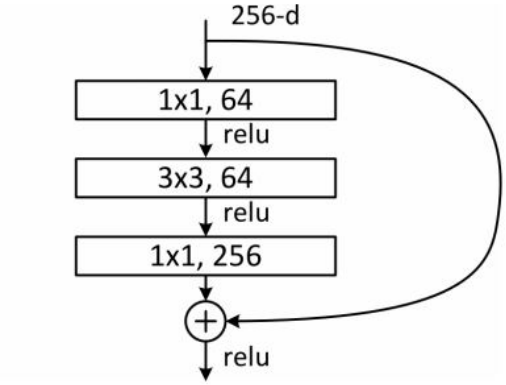
Obrázek 3.2: Modul Inception, který reprezentuje základní stavební jednotku architektury GoogLeNet [61].

3.1.5 ResNet

Síť ResNet, navržená výzkumnou skupinou Microsoft Research [24], byla mnohem hlubší něž ostatní architektury. Síť ResNet obsahovala 152 vrstev a zvítězila v ILSVRC 2015 s mírou chybovosti neuvěřitelných 3.57 %, což je nižší chybovost, něž dosahují lidé (lidská chybovost je mezi pěti až deseti procenty odvíjející se od zkušeností řešitele [27]). Ačkoli je hloubka sítě v té době inovativním řešením, nejednalo se o nejinovativnější věc v této architektuře. Nejinovativnější věcí této architektury bylo zavedení struktury nazývané *residual bock* [24].

V roce 2011, Pierre Sermanet a Zann Lecun, představili toto řešení v práci [55] a označili ho jako *bypass* vrstvy. V architektuře ResNet se autoři tímto řešením inspirovali, ale zašli ještě dál. Místo vynechání jedné vrstvy, obešli vrstvy dvě, jak je znázorněno na obrázku 3.3. V klasických konvolučních neuronových sítích je počítána pouze F(x). To znamená, že další vrstva pracuje pouze s tímto výsledkem a nemá přímé propojení s originálním vstupem. Pokud se obejdou dvě vrstvy a tento vstup se následně přidá do následující transformace

F(x), jsou získány mapy vlastností reprezentují změny v tomto okamžiku. Autoři architektury uvádějí, že je jednodušší optimalizovat toto odkazované mapování místo optimalizace parametrů, na které není odkazováno vůbec. Autoři experimentovali s architekturami obsahujícími 18, 34, 50, 101 a 152 vrstev. Během těchto experimentů narazili na problém s počtem parametrů v hlubších vrstvách sítě. Autoři však tento problém vyřešili obdobným způsobem jako v architektuře GoogLeNet a redukovali tak počet vlastností na přibližně jednu čtvrtinu využitím konvoluční vrstvy o velikosti 1 x 1. Tento mechanismus byl implementován do ResNet-50 a více.



Obrázek 3.3: Schéma struktury residual-lock využívané v architektuře ResNet [24].

3.2 Architektury a detektory konvolučních neuronových sítí

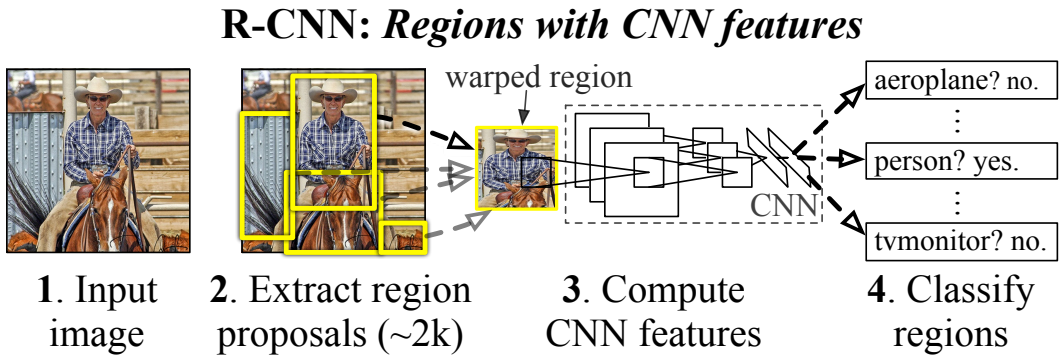
3.2.1 R-CNN

Jedná se o detektor založený na regionech. Autoři článku [20] navrhují R-CNN detektor, který lze použít v úkolech detekce objektů. Jejich práce patří mezi první, která ukazuje, že CNN by mohl vést k výrazně vyššímu výkonu detekce objektů na datových sadách PASCAL VOC než u systémů založených na jednodušších funkcích podobných HOG. Hluboká metoda učení je ověřena jako účinná a efektivní metoda v oblasti detekce objektů.

Detektor R-CNN se skládá ze čtyř modulů zobrazených v obrázku 3.4 a popsaných dále. První modul nejprve vyhledává oblasti, kde se hledaný objekt pravděpodobně nachází. Tyto oblasti jsou nezávislé na dané klasifikační třídě, do které objekt pravděpodobně spadá.

Pro vyhledávání oblastí je využíván algoritmus selektivního vyhledávání, který kombinuje metodu Kompletního vyhledávání a segmentace [63]. Z kompletního vyhledávání se snaží zachytit všechna možná umístění objektu a ze segmentace je využívána myšlenka sledovat strukturu obrazu. Algoritmus tedy prochází vstupní obraz a snaží se identifikovat podobné pixely, ze kterých se může skládat sledovaný objekt.

Následně je využita konvoluční neuronová síť složená z pěti konvolučních vrstev a dvou plně propojených vrstev pro extrakci vektorů vlastností z každé navrhované oblasti. Jelikož plně propojené vrstvy vyžadují vstup pevné velikosti, měl by mít vektor vlastností stejnou velikost. Autoři se proto rozhodli stanovit fixní velikost na 227 x 227 pixelů jako vstupní velikost pro konvoluční neuronovou síť. Bez ohledu na velikost a poměr stran byly navr-



Obrázek 3.4: 1. Vstupní obraz, 2. extrakce přibližně 2000 potencionálních oblastí na výskyt objektu, 3. deformace regionů na rozměr 227 x 227 pixelů a výpočet vektoru vlastností pro každý obraz pomocí konvoluční neuronové sítě. 4 klasifikace každého regionu pomocí SVM pro specifickou třídu [20].

hované oblasti deformovány na pevně stanovenou velikost 227 x 227 pixelů a předány jako vstup do konvoluční neuronové sítě.

Extrahované vlastnosti z výstupu sítě jsou předány třetímu modulu složeného z SVM klasifikátorů, školených pro každou třídu zvlášť, které jsou zodpovědné za rozhodnutí, zda se sledovaný objekt v dané oblasti nachází a do které sledované třídy spadá. Poslední modul zodpovídá za výpočet a vykreslení ohraničujícího boxu sledovaného objektu. Obecně má ohraničující rámeček velké překrytí s pozadím. Toto je poslední problém, se kterým se musí algoritmus vypořádat. Pro zlepšení lokalizace (zpřísnění ohraničujících rámečků) se používá lineární ohraničující regresní metoda navržená v práci [15] s jednou úpravou. R-CNN aplikuje regresi na prvky vypočtené neuronovou sítí, namísto na geometrické prvky vypočtené pomocí deformovatelném modelu.

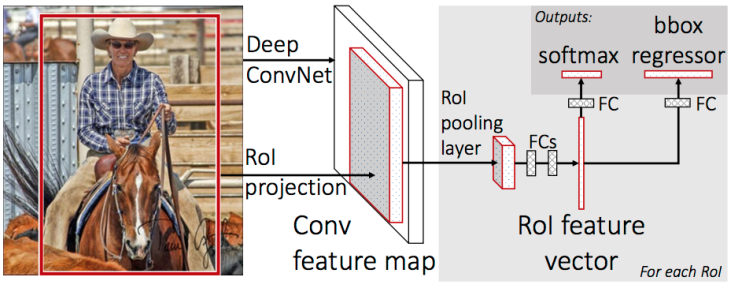
Ačkoliv R-CNN překonal svou rychlostí podobné architektury, patří právě rychlost k jeho největším nedostatkům. Rychlost omezují převážně tyto prvky:

- Dopředná propagace přes CNN (každá oblast každého obrazu musí být předána samostatně),
- jeho trojnásobným školením (síť pro generování vektorů vlastností, síť pro rozhodnutí o třídě a regresní model pro ohraničování objektů),
- generování potencionálních oblastí výskytu objektu.

3.2.2 Fast R-CNN

Krátce po detektoru R-CNN byl autory Rossemn Girshickem a kol. [19] představen detektor Fast R-CNN, který přináší podstatné zlepšení oproti originálnímu řešení.

Prvním vylepšením bylo odstranění samostatných výpočtů vlastností konvolučních sítí pro každý navrhovaný region, a to tak, že je nejprve získána mapa vlastností z celého obrazu, až poté dochází k vytváření navrhovaných regionů. Navrhované regiony jsou extrahovány z mapy vlastností namísto vstupního obrazu. Díky tomu, že je mapa funkcí sdílena pro všechny navrhované regiony, dochází ke snížení paměťových nároků a úspoře času. Pro každou navrhovanou oblast je z mapy vlastností získán vektor pevné velikosti pomocí algoritmu nazývaného RoI Pooling.



Obrázek 3.5: Architektura Fast R-CNN. Vstupní obraz a více zájmových oblastí jsou vstupem do plně propojené konvoluční sítě. Každý RoI je sdružen do mapy prvků s pevnou velikostí a poté mapován do vektoru prvků pomocí plně propojených vrstev. Síť má dva výstupní vektory pro každou oblast zájmů: pravděpodobnosti softmaxu a posun ohraničujícího boxu pro každou třídu [19].

Region of Interest (RoI) neboli sdružovací algoritmus, který bere souřadnice oblastí získaných pomocí selektivního vyhledávání a přímo je ořezává z mapy vlastností původního obrázku. Sdružování oblastí zájmů umožňuje sdílení výpočtů pro všechny regiony, protože konvoluční síť nemusí provádět výpočet pro každou oblast. Výpočet konvoluční neuronovou sítí probíhá pouze jednou, aby ze vstupního obrazu mohl vygenerovat jedinou funkční mapu. Oříznutím této mapy vlastností se vypočítají vlastnosti pro různé regiony.

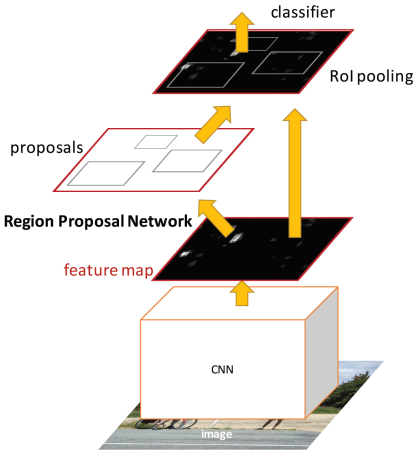
Fast R-CCNN také spojil oddělené moduly pro extrakci vlastností, klasifikaci a regresi do jednoho kroku. Místo oddělených SVM klasifikátorů pro každou třídu byla přidána softmax vrstva a regresní paralelní vrstvy pro ohraničující boxy na konci modelu, které jsou tímto způsobem trénovány současně. Tyto změny vedly ke snížení času a paměťových nároku potřebných k trénování. Časová náročnost klesla z původních 84 hodin na 9 hodin.

3.2.3 Faster R-CNN

Třetím vylepšením původního modelu R-CNN a jeho nástupce Fast R-CNN, které vyřešilo časovou náročnost selektivního vyhledávání, byl představen v roce 2016 v článku [52].

Tým Microsoft výzkumníků zjistil, že mapa vlastností počítaná v prvním části modelu Fast R-CNN může být využita pro generování navrhovaných regionů místo pomalého algoritmu selektivního vyhledávání. Algoritmus selektivního vyhledávání byl v tomto detektoru nahrazen sítí regionálních návrhů (RPN). RPN muselo být schopné předpovídat oblasti s různým rozlišením a poměrem stran. Tato schopnost byla zajištěna tím, že autoři využívají nového konceptu kotevních ohraničení velikostí 32 px, 64 px a 128 px a poměru stran $1:1$, $1:2$ a $2:1$, což vede k 9-ti různým typům kotevních ohraničení. Jakmile se rozhodne o umístění v obrázku, je těchto 9 ohraničení ořízнuto z tohoto umístění. Ohraničení jsou ořízнuty po daném počtu pixelů, který určuje kompresní faktor. Proces začíná vlevo nahoře a končí v pravé dolní části obrázku, proto ho v některé literatuře nazývají jako posuvné okno.

Jak je patrné z obrázku 3.6 proces velmi obdobný předchozímu modelu Fast R-CNN, kdy následuje RoI sdružování a následná klasifikace objektu.



Obrázek 3.6: Faster R-CNN je jednoduchá jednocestná síť pro detekci objektů. Vstupní obraz je nejprve zpracován konvoluční neuronovou sítí a výstupní mapy vlastností jsou předány ke zpravování RPN. Po získání požadovaných oblastí je provedena klasifikace [52].

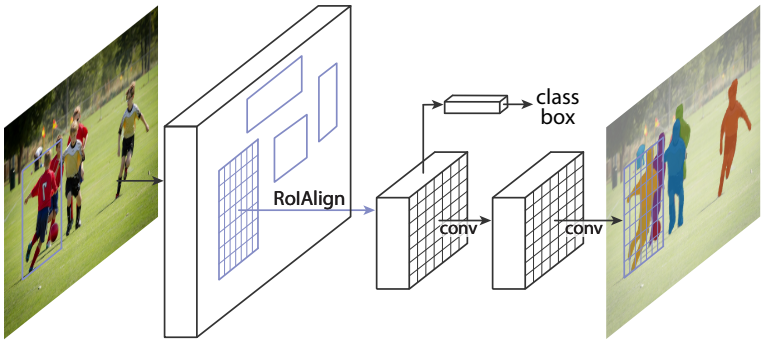
3.2.4 Mask R-CNN

Mask R-CNN je dalším rozšířením již představeného modelu Faster R-CNN. Zmíněný model Mask R-CNN byl navržený v roce 2017 společností Facebook AI Research FAIR [23]. Model rozšiřuje Faster R-CNN o paralelní větev pro predikci masky objektu. Na základě této predikce je generována kvalitní segmentační maska pro každou instanci nalezeného objektu (Object Instance Segmentation).

Segmentace každé instance je náročný proces, který vyžaduje správnou detekci všech objektů v obraze a také jeho přesnou segmentaci. Kombinuje proto prvky z klasického počítačového vidění a sémantickou segmentaci. Počítačové vidění využívané při detekci objektů, kde je cílem klasifikovat jednotlivé objekty a lokalizovat každý nalezený objekt pomocí rozhraničovacího rámečku. Sémantická segmentace klasifikuje každý pixel do pevně dané sady kategorií.

Metoda Mask R-CNN rozšiřuje Faster R-CNN přidáním paralelní větve pro predikci segmentačních masek v každé oblasti zájmů (RoI). Tato větev je v podstatě FCN aplikovaná na každý nalezený objekt zájmů, která předpovídá segmentační masku způsobem nazývaný pixel-to-pixel. Tato větev představuje pouze nepatrnou výpočetní režii.

Klíčovou roli v predikci masky představuje správné zarovnání oblastí z původního obrazu a obrazu výstupního. Jelikož původní metoda RoIPool při kvantifikaci oblastí využívá nejčastěji operaci Max Pooling, následně provede zaokrouhlování, dochází k chybám v zarovnávání mezi jednotlivými oblastmi zájmu a extrahovanými vlastnostmi. I když tyto chyby nemusejí mít velký vliv na klasifikaci, mají velký negativní vliv na predikci masky s přesností na pixel. Autoři tento problém vyřešili zavedením nové metody nazývané *RoIAlign*, která je znázorněna na obrázku 3.7. Metoda RoIAlign odstraňuje kvantizaci RoIPool a místo toho používá bilineární interpolaci pro výpočet přesných hodnot vstupních funkcí na čtyřech pravidelně vzorkovaných místech v každém RoI a následné agregaci výsledků. Zavedením RoIAlign dochází k relativnímu zpřesnění masky o 10 % až 50 %, což má za následek přesnější zarovnání pixelů z původního obrazu a pixelů z extrahovaných vlastností. Koncovým mechanismem je kombinace vygenerovaných masek s predikovanými hraničními oblastmi. Metoda vykazuje velmi přesnou predikci tříd společně s jejich přesnou lokalizací pomocí segmentační masky.



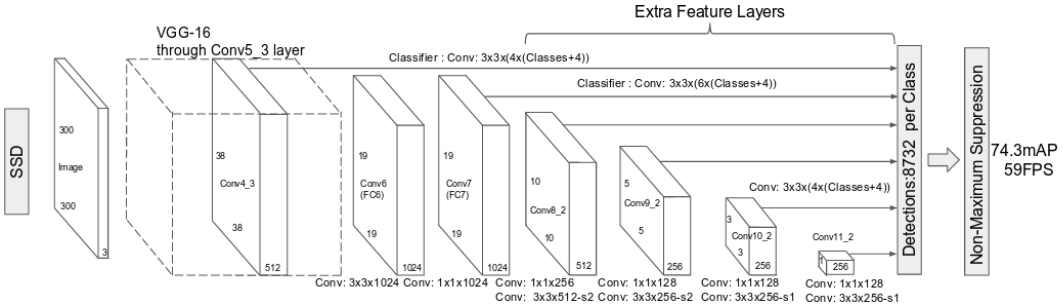
Obrázek 3.7: Obrázek znázorňuje architekturu modelu Mask R-CNN s využitím metody RoIAlign [23].

3.2.5 Single Shot MultiBox Detector

Jádro Single Shot Multibox Detector (SSD)[36] predikuje skóre jednotlivých tříd a relativní posunutí boxů pro pevnou sadu výchozích ohraničení pomocí malých konvolučních filtrů aplikovaných na mapy vlastností. Tento model je první představený detektor založený na dopředné konvoluční neuronové sítí, který při své činnosti eliminuje počet návrhů na ohraničení objektů a nevykonává převzorkování pixelů. To má za následek významné zvýšení rychlosti se zachováním vysoké přesnosti detekce.

Síť znázorněná na obrázku 3.8 nejprve vytvoří kolekci ohraničení pevné velikosti a skóre, které udává pravděpodobnost výskytu konkrétní třídy v těchto ohraničeních. Následuje krok non-maximum potlačení, jež vytvoří konečnou detekci. Struktura sítě se většinou rozděluje do dvou části. První část sítě je založena na standardní struktuře využívané pro klasifikaci obrazů. Druhou část sítě tvoří postupně se zmenšující konvoluční vrstvy, které umožňují predikci objektů různých velikostí. Použitý model se liší od ostatních detekčních modelů tím, že umožňuje detekci na různých vrstvách, a tak zvyšuje celkový počet nalezených objektů.

Každá vrstva může generovat pevně daný počet predikcí pomocí sady konvolučních filtrů. Pro vrstvu o velikosti m x n je základním elementem pro predikci potencionálních detekcí malé jádro o velikosti 3 x 3, které udává buď pravděpodobnosti pro výskyt dané třídy, nebo relativní posunutí tvaru vzhledem k výchozímu poli souřadnic výchozího ohraničení. Mapy příznaků jsou rozděleny do jednotlivých buněk, kde se v každé buňce zkoumá



Obrázek 3.8: Architektura modelu SSD, který na konec základní sítě přidává několik vrstev, které jsou schopné predikovat posuvy od výchozích ohraničení v různých měřítkách a poměrech stran [36].

relativní posunutí oproti výchozímu ohraničení. Stejně tak je získáno skóre pro každou třídu, které udává, zda se v dané výseči nachází objekt dané třídy.

Tyto konstrukční funkce vedou k jednoduchému úplnému zaškolení a vysoké přesnosti, a to i na vstupních obrázcích s nízkým rozlišením, což dále zlepšuje kompromis mezi rychlostí a přesností. Autoři detekčního modelu SSD udávají, že je model schopen provádět detekci na 59 snímcích za sekundu s přesností mAP 47.3 % pro model SSD a 22 snímků za sekundu s přesností mAP 76.8 % pro model SSD512. Dle uvedených výsledků přináší tato metoda výrazné zrychlení proti metodě Faster R-CNN a YOLO implementované na stejné architektuře VGG16.

3.2.6 RetinaNet

Model RetinaNet je jednostupňový detektor, který jako první detektor překonal přesnost dvoustupňových detektorů při zachování rychlosti [33]. K dosažení těchto výsledků přivedlo autory článku identifikování zásadního problému, který limituje přesnost jednostupňových detektorů v porovnání s detektory dvoustupňovými. Tímto problémem je třídní nerovnováha(*class imbalance*), se kterou se detektory během tréningové fáze setkávají. Model RetinaNet se snaží tento problém vyřešit zavedením nové ztrátové funkce s názvem *focal loss*.

U dvou stupňových detektorů jsou v první fázi vyhledávány oblasti, ve kterých se pravděpodobně nahází hledaný objekt. Tyto oblasti jsou za pomoci nejrůznějších kaskád a vzorkovacích heuristik značně redukována typicky na 1000-2000 oblastí. Díky tomu jsou z velké většiny zredukovány oblasti, které obsahují pouze pozadí a nikoliv sledovaný objekt. V druhé klasifikační fázi je pomocí heuristik, jako *fixed foreground-to-background ratio*, nebo *hard example mining* [56], vytvořena třídní rovnováha mezi popředím a pozadím.

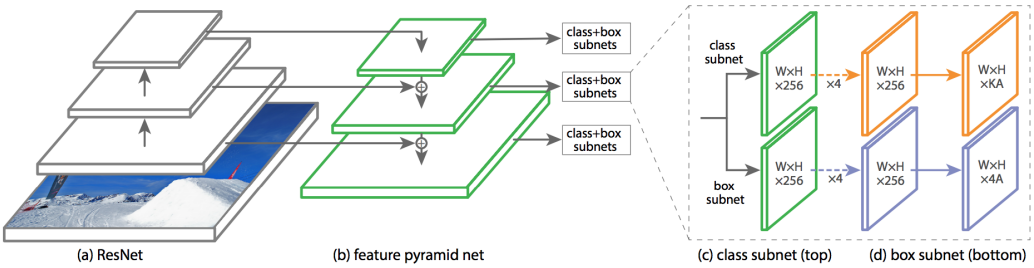
Avšak u jednostupňových detektorů je situace složitější. Jelikož tyto detektory postrádají první fázi a tudíž nejsou schopny dostatečně zredukovat predikované oblasti, musí běžně zpracovat i stovky tisíc oblastí. Tento proces doprovází třídní nerovnováha, která je způsobena tím, že detektor v jednom kroku hustě vzorkuje oblasti z celého obrazu. Avšak oblast, na které se nachází sledovaný objekt, zpravidla nevyplňuje převážnou část obrazu a proto většina těchto vzorkovaných oblastí odpovídá pozadí a nikoli sledovanému objektu. Vysoký počet oblastí obsahující pouze pozadí způsobuje neefektivní trénování detektoru. Druhým problém spojeným s třídní nerovnováhou je skutečnost, že snadné detekce mohou vézt k celkové degeneraci modelu. K eliminaci tohoto problému se běžně využívají tech-

niky jako například *hard example mining*. Avšak použití zmíněných technik bývá neúčinné, jelikož v tréninkových datech stále dominují snadno klasifikované oblasti obsahující pozadí.

Autoři článku představili již zmiňovanou funkci *Focal Loss*, která je efektivní alternativou první fáze dvoustupňových detektorů a eliminuje tak celkovou třídní nerovnováhu. Představená ztrátová funkce je dynamicky škálovatelná, která se zvyšující se důvěrou ve správnost třídy snižuje škálovatelný faktor k nule. Díky tomu může funkce intuitivně snižovat váhu pro snadno klasifikovatelné oblasti, které odpovídají pozadí a umožnit tak modelu zaměřit se na náročné detekce odpovídající daným objektům.

Pro demonstraci představeného modelu byla vytvořena síť RetinaNet (obrázek 3.9). RetinaNet - jediná sjednocená síť složená z páteřní sítě a dvou podsítí specifických pro jednotlivé úkoly. Páteřní síť zodpovídá za výpočet mapy konvolučních funkcí na celém vstupním obrazu. První podsíť klasifikuje objekty na výstupu páteřní sítě. Druhá podsíť provádí regresi ohraničujícího boxu. Tyto dvě podsítě jsou navrženy speciálně pro jednostupňovou hustou detekci.

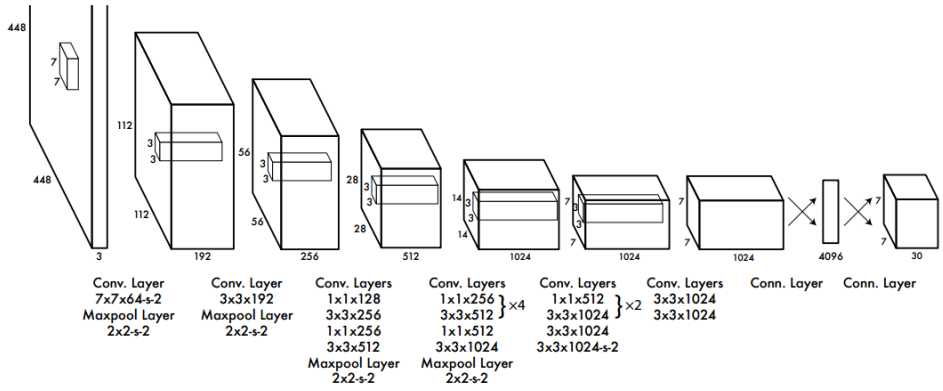
Jako páteřní síť byla použita Feature Pyramid Network (FPN) [32]. Síť FPN rozšiřuje standardní konvoluční síť s cestou shora dolů a postranními spoji a účinně tak ze vstupního obrazu konstruuje vícerozměrnou pyramidu. Každá úroveň pyramidy může být využita pro detekci objektu v jiném měřítku. Podobně jako v článku představující síť FPN [32] jsou i zde využívány kotevní boxy. Na každé úrovni pyramidy se využívají kotvy o třech poměrech. Navíc pro pokrytí hustších měřítek jsou pro každou úroveň přidány další tři sady kotevních boxů, což pokrývá rozsah měřítka 32-813 pixelů vzhledem k vstupnímu obrazu sítě.



Obrázek 3.9: Architektura sítě RetinaNet používá Feature Pyramid Network (FPN), která má na svém vstupu výstup páteřní sítě ResNet. Na obrázku (a) je znázorněna páteřní síť zodpovědná za výpočet mapy vlastností v celém obraze. Obrázek (b) znázorňuje funkci Pyramid Network, na jejichž výstup jsou připojeny dvě podsítě. První síť pro klasifikaci kotevních boxů obrázek (c) a druhou pro regresi ohraničujících boxů obrázek (d) [33].

3.2.7 You Only Look Once

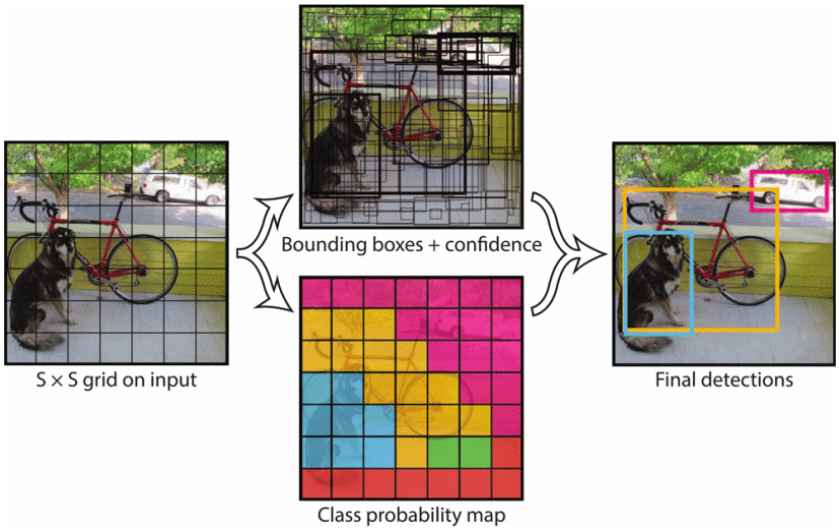
Jak již název napovídá, základní myšlenkou tohoto modelu je podívat se na vstupní obraz pouze jednou [49]. V tomto jediném kroku model předpovídá, kde a jaké objekty se v obrazu nacházejí (znázorněno na obrázku 3.10). Model se skládá pouze z jediné konvoluční neuronové sítě, která předpovídá několik potenciálních objektů a zároveň zajišťuje klasifikaci těchto objektů. Jelikož autoři přistupují k detekci jako k problému s regresí, není potřeba vykonávat výpočetně náročné operace. To má za následek významné zvýšení rychlosti oproti dvoufázovým modelům představeným výše. Autoři článku uvádějí rychlost zpracování 45 snímků za sekundu na GPU Titan X, což umožňuje detekci v reálném čase. Autoři modelu rovněž uvádějí, že model YOLO dosahuje více než dvojnásobku průměrné přesnosti jiných systémů pracujících v reálném čase.



Obrázek 3.10: Detekční síť má 24 konvolučních vrstev následovaných dvěma plně spojenými vrstvami. Střídavě 1 × 1 konvoluční vrstvy zmenšují počet vlastností z předešlých vrstev [49].

Další předností modelu YOLO je skutečnost, že na rozdíl od modelů založených na posuvném okně nebo techniky region-preposal vidí během trénovací a testovací fáze celý obraz. Díky tomu model zvládá pracovat i s kontextuální informací objektu. Tato skutečnost přispívá k celkové robustnosti modelu.

Jak je znázorněno na obrázku 3.11, vstupní obraz je nejprve rozdělen do mřížky velikosti S x S. V případě, že se střed objektu nachází v buňce mřížky, je tato buňka zodpovědná za detekci tohoto objektu. Každá buňka mřížky následně předpovídá několik oblastí a skóre spolehlivosti pro tyto oblasti. Skóre spolehlivosti odráží, jak moc je model přesvědčený, že daná oblast obsahuje daný objekt. V případě, že daná oblast neobsahuje objekt, je skóre spolehlivosti rovné nule. V opačném případě se skóre spolehlivosti rovná průniku a sjednocení (*Intersection over Union* - IOU) mezi predikovanou oblastí a skutečnou oblastí. Každá předpovídaná oblast se skládá z 5 předpovědí: x, y, w, h a skóre důvěry. Souřadnice x a y představují relativní střed vzhledem k mezím buňky. Šířka (w) a výška (h) se předpovídají vzhledem k celému obrazu. Skóre důvěry je vyjádřeno pomocí IOU.



Obrázek 3.11: Vstupní obraz je rozdělen do mřížky S x S a pro každou buňku mřížky predikuje B ohraničených oblastí, skóre důvěryhodnosti pro tyto oblasti a C udávající počet predikčních tříd [49].

Každá buňka mřížky také předpovídá pravděpodobnost třídy (podmíněny buňkou mřížky obsahující objekt). Pro každou buňku mřížky je předpovídána pouze jedna sada predikcí třídy bez ohledu na počet generovaných tříd. Architektura sítě vychází z modelu GoogLeNet. Síť je složena z 24 konvolučních vrstev následovaných dvěma plně propojenými vrstvami. Původní počáteční vrstvy využívané v modelu GoogLeNet jsou nahrazeny 1 x 1 redukční vrstvou nasledované 3 x 3 konvoluční vrstvou.

YOLO9000

Model YOLO900 [50] představuje vylepšenou verzi původního modelu YOLO. Původní model YOLO trpěl řadou nedostatků v porovnání s nejmodernějšími detekčními systémy. Analýza chyb modelu YOLO ve srovnání s Faster R-CNN ukazuje, že se model dopouští vysokého množství lokalizačních chyb. Hlavním záměrem autorů bylo zlepšit přesnost sítě při zachování jeho rychlosti. Pro dosažení vyšší přesnosti prošla síť několika zásadními úpravami. První úpravou bylo zavedení dávkové normalizace (*batch normalization*) na všech konvolučních vrstvách, které významně přispívá ke konvergenci modelu. Díky tomu bylo možné z modelu odstranit všechny dropout metody bez nežádoucího přetrénování modelu. Druhé vylepšení spočívá v použití klasifikátoru s vyšším rozlišením. Originální YOLO trénuje síť klasifikátorů s rozlišením 224 x 224 pixelů a pro detekci zvyšuje rozlišení na 448 pixelů. To znamená, že se musí síť přepnout pro detekci učících se objektů a přizpůsobit se novému vstupnímu rozlišení. Tento nedostatek vyřešili autoři v modelu YOLO9000 použitím klasifikátoru s rozlišením 448 x 448 pixelů pro 10 epoch na datovém souboru ImageNet. Díky tomu síť lépe pracuje se vstupními obrazy ve vyšším rozlišením.

Dále byly z modelu odstraněny plně propojené vrstvy, a pro předpovídání potenciónálních oblastí jsou použity, obdobně jako v modelu Faster R-CNN, kotevní boxy (*anchor boxes*). Rovněž byla odstraněna jedna sdružovací vrstva, aby se zvýšilo rozlišení konvolučních vrstev sítě. Dále je upravena síť tak, aby pracovala se vstupním obrazem s rozlišením 416 pixelů namísto původních 448 x 448. Důvodem tohoto rozhodnutí je požadavek na liché

hodnoty buněk v mapě vlastností a definovat tak středovou buňku. Jelikož autoři článku předpokládají, že objekty, obzvláště ty velké, mají tendenci se nacházet ve středu obrazu. Díky tomuto lichému počtu buněk mřížky bude středová oblast predikovat tyto objekty. Namísto varianty s použitím sudého počtu buněk, kde by tyto objekty předpovídali čtyři místa poblíž středu. Konvoluční vrstvy zmenší vstupní obraz s faktrorem 32, a díky tomu je získána mapa vlastností s rozlišením 13 x 13. Jelikož model YOLO9000 využívá pouze konvoluční a sdružovací vrstvy, může být změna velikosti obrazu měněna za běhu. Díky této vlastnosti není vstupní velikost obrazu sítě pevně stanovena, ale je měněna po několika iteracích. Po každých 10 dávkách je náhodně zvolena nová velikost vstupních obrazů. Tento režim nutí síť, aby se naučila dobře předpovídat objekty v různých rozlišeních.

YOLOv3

S třetí verzí modelu YOLOv3 [51] přichází několik vylepšení stávajícího modelu YOLO9000. Změnou prošla predikce ohraničujících oblastí. Skóre objektivity je pro každé ohraničení vypočítáno na základě dvou hlavních kritérií. První kritérium udává, že pokud jedno ohraničení překrývá objekt více než ohraničení předchozí, je vybráno právě toto ohraničení. Druhé kritérium udává, že pokud není předchozí ohraničení lepší, ale překrývá objekt o více než stanovený práh, je predikce ignorována. YOLOv3 predikuje boxy ve třech různých rozlišeních, ze kterých následně extrahuje vlastnosti pomocí konceptu založeném na **Feature Pyramid Networks**. Kvůli tomu bylo přidáno několik konvolučních vrstev.

Jelikož může každá ohraničená oblast obsahovat několik klasifikačních tříd, došlo k odstranění vrstev softmax, které byly shledány jako zbytečné pro zachování dobrého výkonu. Na místo vrstev softmax se během trénovací fáze využívají binární *cross-entrophy* funkce. To pomáhá při trénování komplexních datasetů, které mají mnoho překrývajících se anotací (například žena a osoba).

K extrakci vlastností se využívá nová síť s názvem Darknet-53, která obsahuje 53 konvolučních vrstev. Oproti předchozí verzi Darknet-19, která se využívala v modelu YOLO9000, dosahuje mnohem menších rychlostí zpracování, ale je výrazně přesnější. Model se stále trénuje na vstupních obrazech plné velikosti. Trénování probíhá v různých rozlišeních za pomoci normalizačních dávek. Pro trénování a testování se využívá framework Darknet.

Kapitola 4

Návrh aplikace a implementace

Cílem této diplomové práce je navrhnout a implementovat algoritmus pro detekci a rozpoznávání zbraně v obrázku. Tato kapitola stručně popisuje návrh, implementaci aplikací a prostředků, pomocí kterých je tohoto cíle dosaženo.

4.1 Analýza problému

Detekce zbraní ve scéně je v aktuální době velmi diskutovaným tématem. Několik psychologických studií prokázalo, že lidé, kteří mají přístup ke zbrani, mají několikanásobně větší pravděpodobnost spáchání násilného chování [60]. Systém, umožnující včasnou detekci útočníka se zbraní a následné upozornění na tuto skutečnost odpovídající bezpečnostní složky, může mít pozitivní dopad na počet obětí případného incidentu. Pokud by byl navíc tento systém schopen automaticky rozpoznat o jaký druh palné zbraně se jedná, může hrát systém významnou roli v rozhodování, jaký typ bezpečnostních složek k incidentu povolat.

Zásadní vliv na úspěšné detekování a rozpoznání zbraně má bez pochyby kvalita obrazu, umístění a orientace zbraně, a celkové zakrytí zbraně. Jak již bylo popsáno v kapitole 2 existuje celá řada možností, jak požadovaný objekt ve scéně detekovat. Z dříve popsaných možností je nejvíce vhodná detekce zbraní pomocí hloubkového učení. Detekce objektů založená na hloubkovém učení je v současné době velice probíranou a rychle se rozvíjející oblastí. Díky tomu existuje celá řada algoritmů, vyvinutých přímo pro potřeby detekce objektů a zároveň zde vzniká dostatek prostoru pro budoucí vývoj. Z představených základních architektur bylo nutné vybrat jednu, na které bude model natrénován. Při výběru modelu bylo přihlíženo k několika faktorům. První faktor, ke který ovlivnil výběr, je přesnost modelu. Ačkoliv jsou modely založené na R-CNN často využívanými a léty ověřenými modely, jejich přesnost a rychlost již byla překonána novějšími a modernějšími modely. Výjimku z modelů založených na R-CNN představuje model Mask R-CNN, který je velice přesný, ale vyžaduje pro své trénování datový soubor obsahující přesné segmentační masky objektu místo ohraničujících boxů. Což by znemožnilo využití programu pro generování zbraní a vedlo ještě k časově náročnějšímu ručnímu anotovaní objektů. Kvůli přesnosti a rychlosti nebyl vybrán ani model SSD. Modely z rodiny YOLO vykazují sice dobrou rychlost, ale přesností stále vyniká model RetinaNet. Tento model nabízí vysokou přesnost se zachováním dostatečné rychlosti zpracování. Z tohoto důvodu byl vybrán model RetinaNet pro trénování modelu v této diplomové práci.

Jelikož síť RetinaNet spadá do kategorie hloubkového učení, bylo zapotřebí získat dostatečné množství dat, na kterých by bylo možné detekční síť natrénovat. Vzhledem k tomu, že výsledkem práce má být detektor, který je schopný nejen zbraň ve scéně detekovat, ale také ji zařadit do správné kategorie, bylo nutné získat dostatečně velký datový soubor, který obsahuje nejen zbraně, ale také kategorii, do které zbraň spadá. Tento problém se však ukázal jako značně komplikovaný, jelikož se mi nepodařilo nalézt dostatečně velký datový soubor obsahující zbraně a ještě kategorizované dle potřeby této diplomové práce. Další možností bylo vytvořit nový datový soubor přesně pro účely této práce. Avšak nasbírat dostatečné množství obrázků zbraní, následně provézt jejich anotaci a poté model trénovat by vyžadovalo obrovskou časovou kapacitu.

Proto byla zkoumána jiná alternativa, jak celkový vývoj urychlit. Po společné konzultaci s vedoucím práce byla nalezena zajímavá alternativa v podobě generování obrázků pomocí 3D modelů zbraní. Výzkumná skupina STRaDe[1] již má k dispozici generátor obrázků, který umožňuje nahrát model ve formátu g3db. Při práci s generátorem jsem však narazil na dva problémy. Pro generování obrázků krátké zbraně fungoval generátor dobře, avšak při načtení komplexního modelu, který byl vytvořen z většího množství polygonů, skončil program chybovým hlášením. K odstranění nalezeného problému bylo nutné zredukovat počet polygonů daného 3D objektu. Tato redukce však vedla ke snížení kvality modelu, což není žádoucí. Druhým nedostatkem, byla absence informace o přesné poloze objektu v obraze pro následný proces anotace. Polohu objektu ve scéně je možné dopočítat, avšak tento proces by byl značně komplikovaný, jelikož by model musel zůstat po celou dobu generování statický a rotace objektu by musela probíhat vždy pouze podle jedné z os. Po prozkoumání všech kladů a záporů byla vytvořena samostatná aplikace, který slouží pro generování obrázků zbraní z 3D modelů včetně velmi přesné polohy objektu ve scéně.

Po úspěšném vygenerování datového souboru přišlo na řadu samotné trénování modelu na základě vygenerovaných dat. Pro samotné trénování neuronové sítě bylo žádoucí nalézt takovou implementaci sítě, která umožňuje trénování modelu na vlastním datovém souboru. Bylo prozkoumáno několik možností a v konečné fázi byla zvolena již existující implementace Keras RetinaNet objekt detektoru vyvinutá autory Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He and Piotr Dollár [18]. Implementace je nejen neustále vyvíjena, ale obsahuje i mnoho pomocných nástrojů pro práci s datovým souborem.

Posledním krokem byla implementace aplikace, která umožňují detekci zbraní na základě natrénovaného modelu. Jelikož je požadováno testování alespoň na 500 obrázcích, bylo vhodné umožnit v aplikaci práci, jak s jednotlivými obrázky zbraní, tak i s celou složkou obrázků. Ačkoliv je zadáním práce testování na statických obrázcích, v reálné situaci může být vstupem do aplikace například krátké video nebo přímo vstup z web kamery zaznamenávající ozbrojený incident. Aby bylo možné využít již implementovaných částí z aplikace pro trénování a také z důvodu multiplatformního využití aplikace, byl zachován programovací jazyk Python3.

[1] https://strade.fit.vutbr.cz/cs/

4.2 Použité vývojové nástroje a technologie

V této části jsou stručně popsány nejpoužívanější nástroje a technologie využívané v rámci diplomové práce.

4.2.1 Visual Studio Code

Visual Studio Code je výkonný a nenáročný editor zdrojových kódu vyvinutý společností Microsoft. Je dostupný pro operační systémy Windows, Linux a MacOS. Editor obsahuje vestavěnou podporu ladění, zvýraznění syntaxe, distribuovaný systém správy verzí Git, inteligentní dokončování kódu a mnoho dalších [40].

Zdrojové kódy editoru jsou zdarma a open-source vydané pod MIT License. Dle průzkumu Stack Overflow 2019 Developer Survey[2] je právě Visual Studio Code nejpopulárnějším editorem pro webové, mobilní a hardwarové vývojáře. Tento editor je oblíbený zejména díky jeho rychlosti, flexibilitě a velkému množství snadno dostupných rozšíření, které značně zvyšují produktivitu.

4.2.2 Python

Python je multiplatformní open-source interpretovaný, objektově orientovaný high-level programovací jazyk s dynamickou sémantikou. První verze jazyku byla vytvořena Nizozemským programátorem Guido van Rossum a vydána v roce 1991. Standardní jazyk Python vyvíjí Python Software Foundation a je implementován v jazyce C. Avšak existuje celá řada dalších implementací jazyka Python, mezi které patří CPython, IronPython, RPython a mnoho dalších [17].

Díky vysoké úrovni datových struktur kombinované s dynamickým typováním je jazyk velmi atraktivní pro rychlý vývoj aplikací, jakož i pro použití jako skriptovací jazyk. Jednoduchá a snadno naučitelná syntaxe jazyku Python zlepšuje čitelnost, a proto zlepšuje celkovou udržitelnost zdrojových kódů. Python podporuje moduly a balíčky, které podporují modularitu programu a opakované použití kódu.

Python lze využít pro celou škálu aplikací od grafických desktopových aplikací po aplikace webové. Python je také velice oblíbený pro vývoj komplexních vědeckých a výpočetních aplikacích. Je vhodný pro datové analýzy a vizualizaci. S rozvojem umělé inteligence a neuronových sítí bylo pro jazyk Python vytvořeno mnoho frameworků zabývajících se právě touto problematikou. Mezi nejznámější frameworky patří TensorFlow, Theano, Sci-kit Learn a mnoho dalších.

4.2.3 TensorFlow

TensorFlow je open source knihovna pro numerické výpočty využívající data-flow grafů. Knihovna TensorFlow spojuje modely, algoritmy strojového učení a hloubkového učení do společného a snadno použitelného celku. Původně byla knihovna vyvinuta týmem Google Brain[3] v rámci výzkumné organizace Machine Intelligence společnosti Google zabývající se strojovým učením a výzkumem hlubokých neuronových sítí. Později se ukázalo, že systém je natolik obecný, že lze využít i v celé řadě dalších domén. První verze byla vydána v únoru 2017 a její vývoj pokračuje doposud [1].

[2] https://insights.stackoverflow.com/survey/2019
[3] https://research.google/teams/brain/

Knihovna TensorFlow je multiplatformní a je možné ji provozovat na celé řadě zařízení, mezi které patří CPU, GPU, mobilní zařízení včetně Embedded zařízení a dokonce i Tensor Processsing Unit(TPU). TPU jsou specializovaná hardwarová zařízení vyvinutá společností Google pro strojové učení pomocí neuronových sítí.

Jednou z největších výhod, kterou TensorFlow poskytuje pro strojové učení, je abstrakce. Místo toho, aby se vývojáři zabývaly drobnými detaily implementačních algoritmů nebo vymýšleli vhodné způsoby, jak připojit výstup jedné funkce ke vstupu jiné, může se vývojář soustředit na celkovou logiku aplikace. TensorFlow se stará o detaily v pozadí.

V říjnu roku 2019 byl vydán TensorFlow 2.0, který byl na základě zpětné vazby uživatelů výrazně přepracován tak, aby byla práce s tímto frameworkem snadnější a výkonnější. Distribuované školení je snadnější provozovat díky novému Keras API . Dále byla přidána podpora pro TensorFlow Lite umožnující nasazení modelů na větší řadu platforem. Kód napsaný pro dřívější verze TensorFlow však musí být přepsán, aby se maximálně využily nové funkce TensorFlow 2.0.

4.2.4 Keras

Jak již bylo zmíněno TensorFlow 2.0 využívá jako primárním TensorFlow API Keras. Keras byl vytvořen tak, aby byl uživatelsky přívětivý, modulární, snadno rozšiřitelný. API bylo navrženo pro člověka, ne pro stroje, dodržuje doporučené postupy pro snižování kognitivní zátěže.

Neuronové vrstvy, ztrátové funkce, inicializační schémata, aktivační funkce a další stavební jednotky neuronových sítí jsou rozděleny na samostatné moduly, které lze kombinovat a vytvářet tak nové modely. Nové moduly napsané v jazyku Python lze snadno přidat jako nové třídy nebo funkce.

Největší důvody, proč využívat Keras, vycházejí z jeho hlavních principů - uživatelská přívětivost. Kromě snadného učení a snadného vytváření modelů nabízí Keras výhody širokého rozšíření, silné uživatelské podpory, integrace s nejméně pěti back-end enginů (TensorFlow, CNTK, Theano, MXNet a PlaidML), a také silnou podporu pro distribuované školení pomocí více GPU. Navíc Keras podporují společnosti jako jsou Google, Microsoft, Amazon, Apple, Nvidia, Uber a dalšími [9].

4.2.5 NVIDIA CUDA

CUDA je paralelní výpočetní platforma a programovací model, díky kterému je používání GPU pro všeobecné účely jednoduché a elegantní. Vývojář stále programuje ve známých jazycích C, C ++, Fortran nebo v některém z dalších podporovaných jazyků, a zahrnuje rozšíření těchto jazyků ve formě několika základních klíčových slov. Tato klíčová slova umožňují vývojáři vyjádřit obrovské množství paralelismu a nasměrovat kompilátor na část aplikace, která je zpracována na GPU.

Dalším pojem představuje NVIDIA CUDA® Deep Neural Network library (cuDNN), což je knihovna pro hloubkové neuronové sítě s GPU akcelerací. Knihovna poskytuje vysoce propracovanou implementaci pro standardní postupy jako jsou dopředná a zpětná konvoluce, sdružování, normalizace a aktivační vrstvy.

Knihovna cuDNN je využívána výzkumníky po celém světě díky její vysoce výkonné akceleraci na GPU. Knihovna umožňuje zaměřit se spíše na školení neuronových sítí a vývoj nových aplikací, než se zabývat laděním výkonu GPU na nízké úrovni. Knihovna cuDNN je využívána celou řadou frameworků pro hloubkové učení, mezi které patří například Caffe, Keras, MATLAB, TensorFlow a mnoho dalších [42].

4.2.6 OpenCV

OpenCV (Open Source Computer Vision Library) je open-source knihovna pro počítačové vidění a strojového učení. OpenCV byla vyvinuta primárně pro aplikace počítačového vidění. Knihovna je napsána v jazyku C++ a vydaná pod licencí BSD, což umožňuje využívat a upravovat zdrojové kódy. OpenCV obsahuje rozhraní pro jazyky C++, Python, MATLAB a další. Podporované platformy jsou Windows, Linux, Android a MacOS. Existuje více než 500 algoritmů a asi 10krát více funkcí, které tyto algoritmy skládají nebo podporují.

Knihovna obsahuje více než 2500 optimalizovaných algoritmů, které zahrnují komplexní sadu klasických i nejmodernějších algoritmů počítačového vidění a strojového učení. Tyto algoritmy lze použít k detekci a rozpoznání tváří, identifikaci objektů, klasifikaci lidských akcí ve videích, sledování pohybu kamer, sledování pohybujících se objektů, extrahování 3D modelů objektů a mnoho dalších. OpenCV má více než 47 tisíc uživatelů a odhadovaný počet stahování přesahující 18 milionů. Knihovna je hojně využívána ve společnostech, výzkumných skupinách a vládních orgánech [45].

4.2.7 Blender

Blender je open-source software, který poskytuje jednu z nejkomplexnějších soustav pro tvorbu 3D grafiky. Blender byl v roce 1998 vyvinutý společností NeoGeo Animation Studios and Not A Number Technologies a je dostupný pro operační systémy Windows, Linux a MacOS. Zahrnuje nástroje pro modelování, stínování, animaci, úpravy videa a mnoho dalších. Blender je napsán v jazycích C, C++ a Python, které je rovněž možné využít pro napsání vlastních uživatelských rozšíření k tomuto programu.

4.3 Tvorba datového souboru

Zde je popsána tvorba datového souboru od fáze hledání modelů zbraní až po finálně vygenerované snímky. Dále je zde stručně popsána knihovna Three.js, která je nedílnou součástí vytvořené aplikace pro generování snímků zbraní z 3D objektů.

4.3.1 Příprava dat

Jak již bylo zmíněno v předchozí části, základní myšlenkou pro tvorbu datového souboru je generování obrázků zbraní na základě 3D modelu zbraně s různým pozadím. Aby bylo možné obrázky generovat, bylo zapotřebí získat 3D modely zbraní. Hlavním parametrem pro výběr modelu byla cena, následně pak kvalita modelu. Při výběru modelu bylo navštíveno několik webových stránek a získáno přibližně 40 modelů zbraní.

Jelikož byla většina získaných modelů zdarma, byla jejich kvalita často nedostatečná nebo neobsahovaly kompletní model včetně textur. V konečné fázi bylo získáno 7 poměrně kvalitních modelů zbraní. Některé z vybraných modelů neobsahovaly pouze model zbraně, ale i další elementy, jako například zásobník nebo nábojnice. Tyto elementy nejsou pro generování datového souboru žádoucí a proto bylo nutné je z modelu odstranit. Jelikož je při generování snímků ze 3D modelů zbraní rotováno, bylo nutné modely vycentrovat a upravit jejich měřítko pro následné usnadnění manipulace v prostředí generátoru.

Úprava modelů byla provedena v grafickém editoru Blender, který je zobrazen na obrázku 4.1. Tento grafický editor rovněž umožňuje export 3D modelů ve formátu *obj*[4], jenž slouží jako vstupní formát pro generátor obrázků. Při exportu modelu je společně se souborem ve formátu *obj* získán soubor ve formátu *mtm*, který ukládá informaci o konkrétních texturách daného 3D modelu a rovněž je využíván programem pro generování obrázků zbraní.

Obrázek 4.1: Obrázek znázorňuje 3D model importovaný do aplikace Blender, který je nutné vycentrovat a odstranit nežádoucí objekty.

Druhým elementem pro generování obrázků zbraní bylo pozadí. Při výběru pozadí byl brán v úvahu pohled z jakého je obrázek zachycen a také zachycené prostředí na snímku. Bylo vybráno celkem 35 různých pozadí, vetšinou z prostředí každodenního života jako jsou ulice, náměstí, nádraží, obchody a restaurace. Na většině snímků jsou zachyceny osoby, které jsou při ozbrojených přepadeních nedílnou součástí.

4.3.2 Three.js

Tato knihovna je využívána v programu pro tvorbu datového souboru za pomoci modelů zbraní ve formátu obj a předem zvoleného pozadí. Samotná knihovna Three.js je napsána v jazyku JavaScript a je určena k použití v prostředí klientských webových aplikací. Z větší části to znamená, že bude na některém zařízení spuštěn na straně klienta – ve webovém prohlížeči. Avšak s technologií jako node.js a další by mohla být knihovna využívána i na straně serveru.

[4] http://paulbourke.net/dataformats/obj/

Většina aplikací využívající knihovnu Three.js zahrnuje 3D grafiku v reálném čase, kde interakce uživatele vede k okamžité vizuální zpětné vazbě. Základním kamenem všech těchto aplikací je 3D matematika. 3D grafiku nelze provést bez matematiky a běžná elektronická zařízení ve výchozím nastavení nerozumí 3D konceptům. Zde přichází na řadu knihovna, která tyto matematické operace abstrahuje, optimalizuje a předává rozhraní na vysoké úrovni. Knihovna Three.js přichází s vlastní matematickou knihovnou se specifickými třídami pro 3D matematiku. Existují samostatné knihovny, které se zabývají pouze touto matematikou, ale s knihovnou Three.js je to jen podmnožina mnohem většího systému. Je velice obtížné spravovat 3D stav prvku *div*, který se pomocí CSS změní na 3D objekt. Aby bylo možné vykreslit na plátno něco, co vypadá jako 3D objekt je vyžadováno hodně logiky. Naštěstí knihovna Three.js toto řeší za nás pomocí jedné metody s názvem *render*. Aby bylo možné vykreslit objekt, je nejprve nutné znát obecnou reprezentaci toho, co je 3D svět. V knihovně Three.js je tento 3D svět reprezentován datovou strukturou THREE.Scene.

THREE.Scene

THREE.Scene která slouží pro popis vzájemných vztahů objektů v tomto 3D světě. Základní datovou strukturou knihovny je Object3D. Jedná se v podstatě o analogii virtuálního Document Object Model (DOM) představující objektově orientovanou reprezentaci XML nebo HTML dokumentu. I zde je využívána stromová struktura vytvářená z uzlů, které se dále rozvětvují. Na rozdíl od klasického DOM stromu, kde lze umisťovat objekty nahoru/dolů nebo doleva/doprava a rotace obvykle probíhá pouze kolem jedné osy, je v 3D scéně mnohem více stupňů volnosti. Pro zobrazení vizuálního snímku požadovaného objektu, jehož stav byl nastaven v tomto stromě, je nutné aktualizovat element *canvas* pomocí fukce render například v některé uživatelské akci nebo pomocí nepřetržité smyčky. Abstrakcí pro vykreslování je zde třída THREE.WebGLRenderer.

THREE.WebGLRenderer

Obrazovky počítačů v dnešní době dosahují vysokých rozlišení. Pokud má obrazovka rozlišení 1920x1080 pixelů obsahuje přibližně dva miliony pixelů. Pokud se má pro každý pixel provézt výpočet, který určí hodnotu pixelů odpovídající dané části objektu, a to vše například 60krát za vteřinu, musí být tento výpočet nesmírně rychlý. Právě pro takové účely byla vyvinuta technologie nazývaná hardwarová akcelerace. Většina počítačů, ale i mobilních telefonů a jiných zařízení má nějaké hardwarové zařízení, které dokáže tyto 3D operace efektivně počítat. Toto hardwarové zařízení se nazývá graphics processing unit (GPU).

GPU se liší od klasického CPU tím, že je vyrobeno pro specifické matematické operace, které probíhají paralelně. Paralelní programování představuje zásadní rozdíly oproti programování například v JavaScriptu. Častý problém je přístup ke společné proměnné z různých vláken. Toto odlišné paradigma znamená, že existuje celý další jazyk nazývaný GLSL. Toto je shader jazyk, který v nějaké podobě existuje v jakémkoli nízkoúrovňovém grafickém API. Knihovna Three.js využívá pro nativní zobrazení interaktivní 3D grafiky JavaScriptové API Web Graphics Library (WebGL). WebGL programy se skládají z obslužného kódu napsaného v JavaScriptu a kódu shaderu, který je vykonáván na grafické kartě.

Všechny tyto nízko úrovňové věci jsou odstíněny a veškerá interakce probíhá přes třídu THREE.WebGLRenderer, který převádí daný 3D objekt na spoustu čísel v paměti GPU. Avšak tento renderer lze využívat i pro 2D objekty dokonce i pro obecné výpočty.

Jádro knihovny obsahuje některé zavaděče pro základní 3D struktury, ale všechny běžné formáty, jako je gltf nebo fbx, jsou samostatné moduly. Three.js se nezajímá, jak jsou dané struktury získány, pokud jsou správně analyzovány a jsou z nich vytvořeny přímo objekty typu THREE.Object. Základní zavaděče jsou velmi obecné, načítají obrázky a soubory přímo do některého ze základních objektů jako jsou objety Material nebo Texture. Zavaděče pro specifické formáty se skládají z těchto stavebních bloků. I když komponenty v některých příkladech Three.js jsou považovány za samozřejmost a mohou se zdát jako součást knihovny Three.js, není to z pravidla pravdou a většinou se jedná o samostatné moduly. Běžným příkladem jsou různé *Orbit Controls*. Mimo prostředí three.js neví, jak zpracovat vstup myši, ani jak použít orbitální logiku na kameru.

4.3.3 Aplikace pro generování obrázků

Pro generování obrázků byla vytvořena webová aplikace, která lze spustit ve většině dnešních webových prohlížečů. Aplikace je napsána v jazyce ECMAScript 6, který přináší výhody v podobě modulů, tříd a mnoho dalších. Pro snadnější vývoj a budoucí rozšíření využívá aplikace správce JavaScriptových balíčků NPM[5] (Node Package Manager). Pro zajištění zpětné kompatibility a dalších optimalizací byl použit Babel compiler[6] společně s nástrojem Webpack[7].

Základní stavební bloky

Základní stavební blok tvořící jádro aplikace představuje již dříve zmíněná knihovna Three.js, která je navržena a optimalizována pro práci s 3D grafikou. Jelikož má aplikace sloužit pro generování obrázků z externích 3D modelů a knihovna Three.js sama o sobě tuto funkcionalitu nepodporuje, bylo nutné použít další zavaděče.

Jedním ze známých formátů pro 3D modely je formát *obj*. Pro tento formát existuje zavaděč *OBJLoader*, který lze do aplikace importovat ze složky examples nainstalovaného balíčku three. Stejným způsobem byl do aplikace importován *MTLLoader* zavaděč pro textury daného 3D objektu. Po úspěšném zavedení objektu včetně jeho textur je možné tento objekt vložit do scény a pracovat s daným objektem stejně jako s jakýmkoliv jiným 3D objektem vytvořeným pomocí vestavěných funkcí knihovny.

Důležitou vlastností, kterou knihovna Three.js ve výchozím stavu nepodporuje, představuje možnost pohybu kamery pomocí myši nebo klávesnice. Což je základní funkcionalita, bez které by manipulace s objektem nebyla příliš uživatelsky přívětivá. Pro podporu této funkcionality byl do aplikace přidán modul *OrbitControls*. Objekt typu OrbitControl vyžaduje při inicializaci referenci na objekt kamery a DOM elementu, do kterého je scéna vykreslována. Po úspěšné inicializaci je možné kamerou hýbat v libovolné ose, jak pomocí klávesnice, tak pomocí myši.

[5] https://www.npmjs.com/
[6] https://babeljs.io/
[7] https://webpack.js.org/

Další možností, jak může uživatel ovlivnit vlastnosti scény je skrze uživatelské rozhraní aplikace. Toto rozhraní je do aplikace importováno za pomoci open source balíčku s názvem *dat.gui* vyvinutého týmem Google Data Arts. Díky tomuto balíčku je velmi snadné vytvořit uživatelské rozhraní, které umožňuje uživatelsky modifikovat JavaScriptové proměnné využívané v aplikaci. Uživatelské rozhraní aplikace nabízí možnosti:

- Výběr předpřipravených modelů,
- výběr pozadí,
- nastavení základního umístění a rotace objektu,
- nastavení osvětlení,
- a několik uživatelských akcí.

Osvětlení

Knihovna Three.js obsahuje několik možností jak osvětlit objekt ve scéně. Na obrázku 4.2 jsou znázorněny základní typy osvětlení. Pro účely generování datového souboru byly využívány dva typy osvětlení. Prvním osvětlením byl typ *Ambient Light*, který osvětluje scénu rovnoměrně ze všech úhlů. Tento typ osvětlení byl využitý při generování základní sady obrázků. Jelikož v reálné scéně nejsou jednotlivé objekty vždy osvětleny rovnoměrně bylo zapotřebí přidat další typ osvětlení.

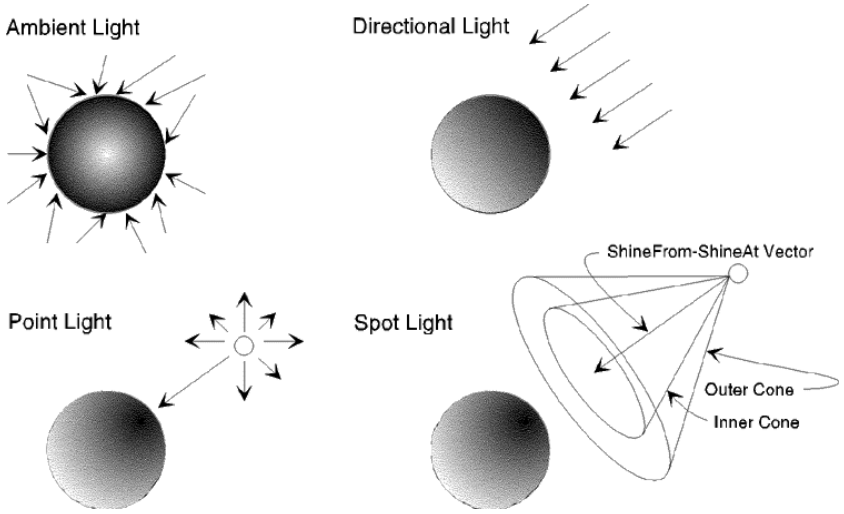
Hlavním kritériem při výběru druhého typu osvětlení byl požadavek na možnost alespoň částečné simulace slunečního světla. Slovo částečné simulace je zde použito záměrně, jelikož v reálné scéně se společně se zbraní nacházejí i jiné předměty. Ostatní předměty mohou mít zásadní vliv na osvětlení daného předmětu a to v podobě překrývání, stínů atd. Při vývoji aplikace byly vyzkoušeny všechny typy osvětlení znázorněné na obrázku 4.2. Z představených typů osvětlení vykazoval na první pohled nejzajímavější výsledky typ *Directional Light*.

Před zahájením, popřípadě i v průběhu generování, je možné u obou typů osvětlení nastavit jeho barvu a intenzitu. Osvětlení typu *Directional Light* umožňuje nastavit další parametry jako například pozici osvětlení.

Výpočet ohraničení objektu

Při tvorbě datového souboru je podstatné nejen zisk dostatečného množství různých obrázků obsahující objekt zájmu, ale je nutné mít také k dispozici polohu sledovaného objektu v tomto obraze. Obrázek obsahující objekt a poloha tohoto objektu společně s třídou, do které objekt spadá jsou tři základní parametry, které budou sloužit jako vstupní parametry při trénování neuronové sítě. Procesu, při kterém se na obrázku ohraničí daný objekt a přidělí třída, do které daný objekt spadá, se nazývá anotace, která bude stručně popsána dále.

Výpočet ohraničení daného objektu probíhá při každém novém vykreslení objektu. Za výpočet ohraničení je zodpovědná funkce z modulu Utils s názvem *computeScreenSpaceBoundingBox* znázorněná na obrázku 4.3, jejíž vstupní parametry jsou: objekt reprezentující ohraničující rámeček, 3D model objektu a instance objektu kamera. Při inicializaci funkce jsou nejprve nastaveny vektory min a max na výchozí hodnoty pro každou osu. Následně jsou procházeny všechny objekty, ze kterých je složen 3D model a kontrolováno, zda se jedná



Obrázek 4.2: Obrázek znázorňuje základní typy osvětlení scény zahrnuté do knihovny Three.js [20].

o objekt typu Mesh. Pokud je objekt typu Mesh, lze z tohoto objektu získat objekt Geometry, který obsahuje informace o poloze, úhlech, barvě a dalších vlastnostech. Z toho objektu je tedy možné získat všechny pozice elementů, ze kterých je objekt složený. Pro všechny takto získané pozice je následně vypočítán vektor odpovídající souřadnicím z pohledu celkové scény s ohledem na aktuální nastavení kamery. Získané souřadnice jsou porovnány s vektory min a max objektu reprezentujícího ohraničující rámeček. Z pozice ohraničujícího boxu jsou následně dopočítány parametry, které slouží jako vstupní parametry pro funkci *download*, a které udávají přesnou polohu objektu zájmu ve scéně.

Jak již bylo zmíněno dříve, poloha sledovaného objektu se vypočítává ze všech elementů, ze kterých je objekt složený. Díky této skutečnosti je vypočítaná poloha objektu velmi přesná a proto je získáno minimální možné ohraničení sledovaného objekt s přesností na 1 pixel.

Anotace objektu

Jak již bylo zmíněno, anotace objektů v obraze je proces, kdy je získána přesná poloha sledovaného objektu v obraze společně s přiřazením třídy, do které sledovaný objekt spadá. Je to základní krok při tvorbě datového souboru určeného pro trénování modelů počítačového vidění. V případě, že jsou objekty v obraze anotovány ručně, je to proces značně časově náročný. Velmi významnou roli zde hraje i přesnost ohraničení daného objektu, která může zásadně ovlivnit celkový výsledek. Existuje několik typů nástrojů, které umožňují obrázky anotovat. Anotace obrazů může probíhat například pomocí specializovaných nástrojů využívajících počítačové vidění, ale v mnoha případech nejsou tyto nástroje dostatečně přesné. Nejčastěji jsou tak obrazy anotovány ručně. Mnoho výzkumných skupin a firem zabývajících se strojovým učením zadávají tuto práci některé ze specializovaných společností, které mají dostatečnou kapacitu zkušených lidí. Pro ruční anotovaní datových souborů je k dispozic celá řada specializovaných nástrojů, mezi ty nejznámější patří LabelImg, LabelMe, Lionbridge AI a mnoho dalších. Existuje několik možností, jak daný objekt v obraze vyznačit v závislosti na použité metodě. Například u modelu Mask R-CNN 3.2.4, který provádí segmentaci požadovaného objektu, jsou objekty anotovány polygonální segmentací.

```
const computeScreenSpaceBoundingBox = (() => {
    const vertex = new THREE.Vector3();
    const min = new THREE.Vector3(1, 1, 1);
    const max = new THREE.Vector3(-1, -1, -1);

    return function computeScreenSpaceBoundingBox(box, mesh, camera) {
        box.set(min, max);
        for (let j = 0; j < mesh.children.length; j++) {
            if (mesh.children[j].isMesh) {
                const position = mesh.children[j].geometry.attributes.position;
                const vector = new THREE.Vector3();
                for (let i = 0, l = position.count; i < l; i++) {
                    vector.fromBufferAttribute(position, i);
                    const vertexWorldCoord =
                        vertex.copy(vector).applyMatrix4(mesh.matrixWorld);
                    const vertexScreenSpace = vertexWorldCoord.project(camera);
                    box.min.min(vertexScreenSpace);
                    box.max.max(vertexScreenSpace);
                }
            }
        }
    }
})();
```

Obrázek 4.3: Obrázek znázorňuje minimální možné ohraničení objektu. Přesná pozice tohoto ohraničení je dostupná v názvu získaného obrázku, která je využívána při procesu anotace objektů.

V tomto případě je objekt anotován pomocí mnohoúhelníku, který umožňuje zachycení objektů s nepravidelnými tvary.

Avšak nejvíce využívanou anotací objektů je anotace pomocí ohraničujících rámečků. Tento způsob anotace byl zvolen i v této práci. Při generování obrázků, je při každém novém vykreslení vypočítáno nejmenší možné ohraničení objektu viz podkapitola 4.3.3. Výsledek výpočtů je předán funkci **download**, která slouží pro stažení obrázku. Vstupní parametry funkce jsou následně přidány do názvu aktuálně staženého obrázku ve formátu:

`x_<hodnotaX>_y_<hodnotaY>_w_<sirka>_h_<vyska>`

kde:

hodnotaX ... udává počet pixelů na x-ové souřadnici od levého horního rohu
hodnotaY ... udává počet pixelů na y-ové souřadnici od levého horního rohu
sirka ... udává šířku modelu v pixelech
vyska ... udává výšku modelu v pixelech

Postup generování

Jak již bylo zmíněno dříve, vývoj aplikace probíhal v editoru Visual Studio Code, který umožňuje otevřít přímo v okně editoru příkazovou řádku, což umožňuje snadnější a rychlejší vývoj aplikace. Aplikaci lze snadno spustit z příkazové řádky zadáním příkazu **npm**

start, čímž dochází ke kompilaci a sestavení programu pomocí nástrojů Babel a Webpack a následnému otevření aplikace ve výchozím internetovém prohlížeči a to na lokální adrese s výchozím portem 8080.

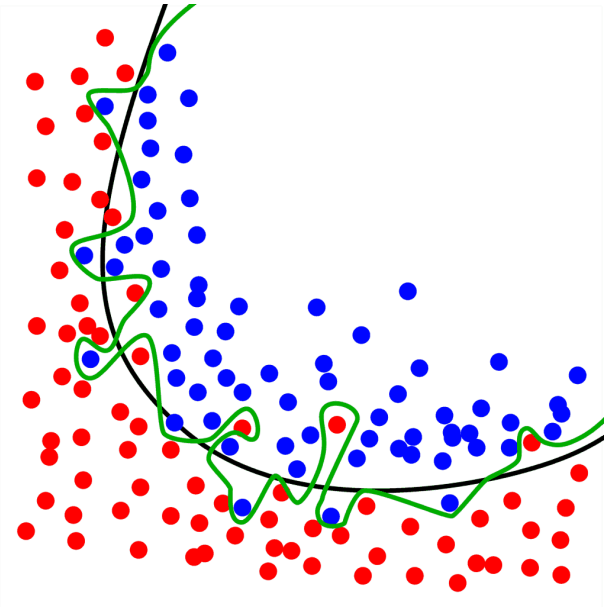
Po úspěšném spuštění aplikace je nejprve nastaveno požadované rozlišení snímků. Rozlišení snímků je závislé na aktuální velikost okna prohlížeče. Toto rozlišení lze snadno nastavit pomoci vývojových nástrojů dostupných po stisku klávesy *F12*, které nabízejí všechny moderní prohlížeče. Po nastavení požadovaného rozlišení je nutné aktualizovat stránku, aby bylo rozlišení aplikováno na scénu. Když je rozlišení nastaveno a scéna aktualizována, přichází na řadu výběr požadovaného modelu a pozadí. Ve chvíli, kdy jsou model a pozadí úspěšně zavedeny, dochází k zobrazení tohoto modelu ve scéně. Před spuštěním samotného generování bylo nastaveno osvětlení scény a počáteční pozice modelu včetně výchozího natočení. Následně byly zvoleny výchozí hodnoty, udávající po kolika stupních se má model skaždou aktualizací pootočit, kolik má být získáno snímků, než bude generování ukončeno a další parametry. V průběhu generování již nebyla pozice modelu manuálně měněna. Na obrázku 4.4 je znázorněn model zbraně Heckler & Koch HK416 před spuštěním procesu generování základní sady.

Obrázek 4.4: Na obrázku je znázorně nastavený model zbraně Heckler & Koch HK416 před samotným spuštěním generování základní sady snímků.

Ve výchozím nastavení je zbraň centrována na střed scény. V první fázi generování zůstává zbraň centrována na střed scény až do chvíle, kdy se celá zbraň otočí kolem osy y. Následně bylo přidáno natáčení i kolem osy x. Po vygenerování této základní sady obrázků se do generování přidávají další parametry, jako automaticky se měnící umístění modelu z pohledu osy x a osy y, a to v závislosti na uplynulém čase a nastavených výchozích hodnotách před samotným generováním. Tyto hodnoty jsou voleny experimentálně a je žádoucí si tyto hodnoty zvolit v závislosti na vybraném modelu, výchozím umístění a velikosti modelu.

4.4 Trénování neuronové sítě

V této části bude stručně popsán proces generování anotací pro vygenerované obrázky až po samotné trénování modelu pomocí knihovny Keras RetinaNet. Při procesu trénování dochází k postupnému zpřesňování modelu na základě vstupních dat. V případě, že je datový soubor malý nebo jsou data velmi podobná, může se po určité době stát, že se model naučí velmi přesně predikovat výsledky v závislosti na datech určených pro trénování. Tento jev je označován jako přetrénování modelu(*overfitting*) znázorněném na obrázku 4.5. Zá-

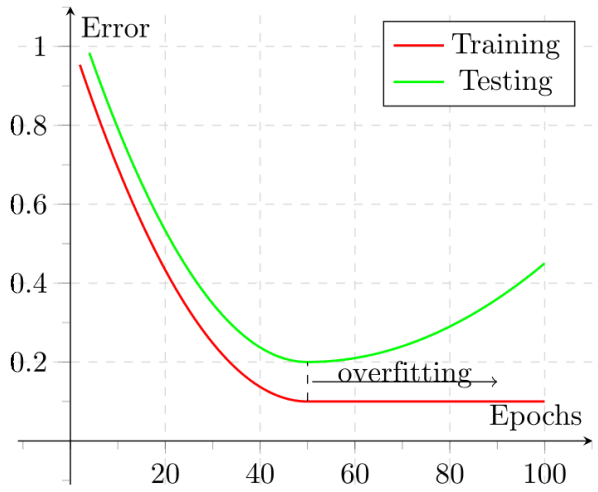


Obrázek 4.5: Černá křivka znázorňuje správně natrénovaný obecný model, který je schopný správné predikce i na nových vstupních datech. Zelená křivka zobrazuje přetrénovaný model, který je schopný velmi přesné predikce na trénovacích datech, avšak na nových datech schopnost správné predikce výrazně klesá [62].

sadním problémem přetrénovaného modelu je skutečnost, že při použití nových vstupních dat postrádá model schopnost dostatečně přesné predikce. V důsledku tohoto jevu ztrácí svou použitelnost v případě, že jsou na vstup vložena nová data. Přetrénování modelu lze v některých případech předcházet snížením počtu iterací, po které je model trénován. Pokud je model trénován pomocí většího počtu iterací dochází rovněž ke ztrátě na obecnosti. V extrémním případě může nastat situace, kdy si model přesně zapamatuje všechna trénovací data a je tak schopný v trénovací fázi dosahovat bezchybných výsledků. Tento stav lze pozorovat v případě, že se chybová funkce blíží k nulové hodnotě, avšak schopnost predikce na testovacích datech se výrazně snižuje. Tento jev je znázorněn na obrázku 4.6. V takovém případě je vhodné trénování modelu ukončit.

4.4.1 Generování anotací

Po úspěšném vygenerování několika tisíc obrázků, bylo nutné pro tyto obrázky vytvořit soubor anotací, které jsou společně se souborem obsahující názvy zkoumaných tříd předkládaný jako vstupní parametry pro trénování modelu. Jelikož knihovna Keras RetinaNet umožňuje trénovat model na vlastním datovém souboru s anotacemi ve formátu csv, byla vytvo-



Obrázek 4.6: Červená křivka znázorňuje míru chybovosti v závislosti na počtu iterací při fázi trénování. Zelená křivka naopak znázorňuje chybovou funkci při testování modelu na testovacích datech. Z grafu je patrné, že ideálním místem pro ukončení trénování je v místě s nejmenší mírou chybovosti při trénovací i testovací fázi [62].

řena jednoduchá aplikace **create-anotation**, která na svém vstupu požaduje název třídy, do které dané snímky spadají a cestu ke složce obsahující vygenerované snímky pro daný model zbraně a vytvoří z nich výsledný csv soubor obsahující požadované anotace.

Vstupními parametry aplikace je tedy adresář s vygenerovanými snímky, pro které má být vytvořen anotovaný soubor. Dalším parametrem je textová reprezentace třídy, do které spadají snímky v daném adresáři. Textová reprezentace třídy musí být shodná s textovou reprezentací třídy v souboru **class.csv**, který je jedním z povinných vstupních parametrů při trénování modelu. Poslední volitelný parametr aplikace reprezentuje název souboru, do kterého mají být anotace přidány. V případě, že nebude tento parametr vyplněn, bude vytvořen nový soubor s výchozím názvem **anotations.csv**.

Každý z anotovaných objektů odpovídá jednomu řádku v souboru anotací. Jednotlivé záznamy jsou ukládány ve formátu:

```
<path>,<x1>,<y1>,<x2>,<y2>,<class>
```

kde:

path ... udává relativní cestu k souboru
$x1$ ... udává umístění začátku ohraničení objektu na x-ové ose
$y1$ ... udává umístění začátku ohraničení objektu na y-ové ose
$x2$ ... udává umístění konce ohraničení objektu na x-ové ose
$y2$ ... udává umístění konce ohraničení objektu na y-ové ose
class... udává textovou reprezentaci názvu třídy daného objektu

Po dokončení fáze generování anotací byly tyto anotace zkontrolovány pomocí aplikace **debug**, jenž je součástí knihovny Keras RetinaNet. Díky této aplikaci je možné před samotnou trénovací fází odhalit nejčastější chyby ve vytvořeném datovém souboru jako například chybná anotace nebo nesprávná cesta k anotovanému snímku. Kontrolu anotací jednotlivých snímků lze vykonat spuštění aplikace příkazem:

```
retinanet-debug csv anotations.csv class.csv
```

4.4.2 Knihovna Karas RetinaNet

Jak již bylo zmíněno výše, pro samotné trénování modlu byla zvolena knihovna Keras RetinaNet objekt detektoru vyvinutá autory Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He and Piotr Dollár [18]. Tato knihovna je jednou z nejlépe hodnocených implementací RetinaNet detektoru na portále GitHub. Knihovna je publikována pod licencí Apache 2.0 [8], která spadá do kategorie svobodného software a umožňuje tak distribuci, úpravy a následnou redistribuci upravené verze software. Knihovna je rozdělena do čtyř hlavních modulů, které budou stručně popsány v této části.

- **train.py** - Tento modul je zodpovědny za trénování modelu. Trénování modelu lze spustit s celou řadou parametrů, které umožňují definovat typ datového souboru, na kterém bude model trénován, počet iterací trénování, počet jednotlivých epoch, soubor po inicializaci vah a mnoho dalších parametrů. Výstupem tohoto modelu je natrénovaný model, který je nutné před samotným procesem detekce zkonvertovat.
- **convert_model.py** - Modul slouží pro převod natrénovaného modelu na výsledný model využívaný v detekční a vyhodnocovací fázi. I tento modul lze spustit s celou řadou parametrů, pomocí kterých lze model specifikovat. Povinné parametry jsou v tomto případě pouze cesta k natrénovanému modelu a cesta, kam má být konvertovaný model uložen.
- **debug.py** - Jak již bylo zmíněno výše, tento modul je vhodné využít před samotnu trénovací fází ke kontrole trénovací datové sady. Tento modul stejně tak jako modul **train.py** nabízí spuštění s různými parametry v závislosti na použitém typu datového souboru.
- **evaluate.py** - Je posledním zmíněným modulem, jehož hlavní funkcí je vyhodnocení úspěšnosti nově natrénovaného modelu. Podobně jako tomu bylo u předešlých modulů, je i tento model možné spustit s celou řadou parametrů v závislosti na použitém typu datového souboru.

4.4.3 Trénování modelu

Pro trénování modelu bylo anotováno celkem 24 226 snímků obsahující vždy pouze jednu konkrétní zbraň. Dalších 3 173 snímků obsahující zbraň bylo anotováno pro potřeby testování vygenerovaného modelu. I když byla vytvořena celkem velká datová sada, byla zde zvolena metoda, kdy je model před-školen na již existujícím a obecnějším datovém souboru a následně je na tomto modelu vykonáno tzv. jemné doladění.

Jemné doladění znamená inicializovat sítě s před-školenými parametry spíše než náhodně nastavenými parametry. Tento způsob je docela populární v modelech založených na CNN kvůli výhodám zahrnující urychlení procesu učení a zlepšení schopnosti zobecnění. Jemné doladění je klíčovou fází pro přizpůsobení modelů konkrétním úkolům a datovým sadám. Obecně platí, že jemné doladění vyžaduje označení třídy pro nový datový soubor školení, který se používá pro výpočet funkce ztráty. V tomto případě budou všechny vrstvy nového modelu inicializovány na základě před-školeného modelu, jako například VGGNet, s výjimkou poslední výstupní vrstvy, která závisí na počtu označených tříd nového datového souboru, a bude proto náhodně inicializována.

[8] https://www.apache.org/licenses/LICENSE-2.0

Jelikož datová sada obsahuje z převážné většiny generované snímky zbraní na základě různých 3D modelů, bylo žádoucí získat základní přehled o použitelnosti této metody v prostředí reálných zbraní. Z tohoto důvodu proběhly tři různé trénování modelu za základě různých vstupních dat.

Všechny tři trénovací fáze probíhaly na páteřní sítí ResNet101, pro prvotní inicializaci vah byl zvolen před-trénovaný model *resnet*101*_oid_v*1.0.0.*h*5[9] trénovaný na datovém souboru Open Images s 500 třídami. Po stažení modelu byl do hlavního adresáře nahrán soubor s anotacemi, soubor s třídami vyskytujícími se v datovém souboru a složka obsahující vytvořený datový soubor. Pro trénování byl využit modul **train** z knihovny Keras RetinaNet, který umožňuje akceleraci výpočtu na GPU.

První trénování bylo zaměřeno pouze na snímky pořízené aplikací pro generování snímků založené na 3D modelu zbraně. Program pro trénování modelu byl spuštěn v celkovém počtu 50 epoch. V rámci jedné epochy probíhalo celkem 5 000 iterací. Během fáze trénování jsou po dokončení jednotlivých epoch ukládány snapshoty, které reprezentují aktuální stav trénování modelu. Po dokončení každé páté epochy bylo provedeno vyhodnocení aktuálně natrénovaného modelu na testovacích datech. Z výsledků testování bylo patrné, že v průběhu dalšího trénování nedochází ke zlepšení přesnosti, ale spíše k jejímu poklesu. Na základě těchto výsledků bylo rozhodnuto o ukončení procesu trénování po 30 epochách. I tak trénování probíhalo přibližně 22 hodin. Výsledky testování byly neuspokojivé, což bude detailněji probráno v kapitole 5.1.

V důsledku velice nízkých výsledků předchozí metody trénování, bylo rozhodnuto o zařazení dalších snímků k vytvořenému datovému souboru. Přidané snímky rozšířily trénovací množinu dat o 2446 snímků krátkých zbraní. Přidané snímky obsahovaly variaci různých typů krátkých reálných zbraní. Na těchto datech proběhly celkem dvě fáze trénování v závislosti na počtu iterací v jednotlivých epochách. Pro obě fáze trénování bylo nastaveno celkem 50 epoch. Avšak během první fáze probíhalo celkem 10 000 iterací v rámci jedné epochy a v druhé fázi byl počet iterací snížen na 5 000. Celkové shrnutí výsledků je popsáno v kapitole 5. Časová náročnost trénování modelu byla pro první fázi přibližně 75 hodin a druhá fáze trvala přibližně 40 hodin.

První pokus o trénování modelu byl spuštěn na notebooku Dell XPS 15 s grafickou kartou Nvidia GTX 1050. Po úspěšné inicializaci modelu začal proces trénování, avšak během prvních několika iterací bylo zřejmé, že použitý hardware není schopen v rozumném časovém horizontu celkový proces dokončit. Potřebný čas k vykonání jedné iterace se pohyboval v rámci jednotek sekund. Po vyzkoušení několika alternativních řešení, která byla rovněž neúspěšná, bylo rozhodnuto o stavbě vlastního dostatečně výkoného počítače, který by byl schopen celý proces trénování dokončit v rozumném čase. V tabulce 4.1 jsou vypsány základní komponenty, ze kterých je složena výsledná počítačová sestava, na které probíhal veškerý další vývoj.

[9] https://github.com/fizyr/keras-retinanet/releases/download/0.5.1/resnet101_oid_v1.0.0.h5

Komponenta	Popis
Operační systém	Windows 10 Pro 64-bit
Procesor	AMD Ryzen Threadripper 1920X 12-Core
Operační paměť	G.SKill TridentZ RGB DDR4 3200 32 GB
Základní deska	ASRock X399 Phantom Gaming 6
Grafická karta	NVIDIA GeForce GTX 1080 Ti
Hardisk	Samsung SSD 970 EVO 1TB

Tabulka 4.1: Obsahuje přehled základních komponent počítačové sestavy využívané k trénování, validaci a testování úspěšnosti natrénovaných modelů.

Během fáze učení je neustále počítána chybová funkce, na základě které lze sledovat celkový postup trénování modelu. Výsledná chybová funkce obsahuje regresní chybu(*regression loss* a klasifikační chybu *classification loss*, která je definována funkcí Focal Loss. Jak již bylo zmíněno v 3.2.6 funkce Focal Loss kombinuje původní funkci Cross Entropy Loss určenou pro binární klasifikaci.

$$CE(p,y) = \begin{cases} -\log(p) & \text{když } y = 1 \\ -\log(1-p) & \text{jinak.} \end{cases} \tag{4.1}$$

kde $y \in \{\pm 1\}$ udává skutečnou třídu a $p \in [0,1]$ označuje jistotu modelem predikované třídy s označením $y = 1$. Pro zjednodušení notace je zde definováno p_t:

$$pt = \begin{cases} p & \text{když } y = 1 \\ 1-p & \text{jinak.} \end{cases} \tag{4.2}$$

po přepsání je funkce $CE(p,y) = CE(p_t) = -\log(p_t)$. Pro kompenzaci třídní nerovnováhy byla definována funkce *Balanced Cross Entropy* definovaná:

$$CE(p_t) = -\alpha_t log(p_t) \tag{4.3}$$

která přidává váhový faktor $\alpha \in [0,1]$ pro třídu 1 a $1-\alpha$ pro třídu -1. Výsledná funkce focal loss je definována:

$$FL(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t) \tag{4.4}$$

kde parametr $\gamma \geq 0$ redukuje relativní ztrátu pro dobře klasifikované příklady.

Konstanty α a γ pro výpočet funkce Focal Loss jsou v knihovně Keras RetinaNet definovány $\alpha = 0.5$ a $\gamma = 2$.

4.5 Implementace aplikace k detekci zbraně

Jedním z bodů zadání této diplomové práce byla implementace aplikace pro detekci a rozpoznání typu zbraně v obrázku. Jelikož je knihovna Keras RetinaNet implementovaná v jazyce Python3, bylo žádoucí tento programovací jazyk zachovat, ať už z hlediska sdílení kódu mezi aplikacemi nebo z důvodů přenositelnosti. Pro účely detekce zbraní na natrénovaném modelu byla rozšířena knihovna Keras RetinaNet o aplikaci **detection**. Obdobně jako pro předešlé aplikace bylo pro vývoj této aplikace využito open-source vývojové prostředí Visual Studio Code. I zde je pro detekci využívána knihovna TensorFlow (verze 2.1.0) a nad touto knihovnou běžící framework Keras (verze 2.3.1). Pro práci s jednotlivými snímky a práci

s videem byla využita knihovna OpenCV (verze 4.2.0). Všechny výše zmíněné knihovny a nástroje jsou multiplatformní a lze je snadno nainstalovat například pomocí správce balíčků Pip.

Vytvořená aplikace umožňuje detekci zbraní z různých zdrojů. Prvním typem zdroje je obrázek nebo složka s obrázky, které jsou postupně načítány a zpracovávány. Druhým typem zdroje může být videozáznam nebo přímo výstup z web-kamery. Video zdroje jsou zpravidla tvořeny sekvencí po sobě jdoucích snímků, kde počet těchto snímků za jednu vteřinu určuje výslednou plynulost pohybujících se objektů. Během zpracování video souborů je video soubor rozložen na jednotlivé snímky, na kterých je dále provedena detekce zbraní. Rychlost zpracování jednotlivých snímků je závislá především na použité metodě, hardware a velikosti snímků, od čehož se odvíjí celková paměťová i časová náročnost zpracování. Ve většině případů není možné snímky zpracovávat v reálném čase. Pro částečnou eliminaci tohoto nedostatku a tedy zvýšení rychlosti zpracování, nabízí aplikace zvolit maximální rozlišení jednotlivých snímků. V případě video záznamu není prováděna detekce na všech snímcích, ale pouze na části z nich v závislosti na použitém parametru při spuštění algoritmu. Počet vynechaných snímků je ve výchozím stavu nastaveno na 4 avšak tento parametr je vhodné volit s ohledem na kvalitu video záznamu a na průměrné rychlosti zpracování jednotlivých snímků na daném zařízení.

Zpracování vstupních snímků je akcelerováno na grafické kartě. V případě, že je na daném hardware k dispozici větší množství grafických karet, je možné specifikovat požadovanou grafickou kartu pomocí vstupního argumentu. Jednotlivé vstupní snímky jsou zpracovány na základě předem natrénovaného modelu ve formátu *.h5*, který ve výchozím nastavení odpovídá modelu trénovaného na základní sítí ResNet101. Při spuštění aplikace je rovněž možné zvolit jiný před-trénovaný model, na základě kterého bude probíhat detekce.

Po úspěšném spuštění aplikace je nejprve načten před-trénovaný model k detekce zbraní. Po načtení zdroje dat jsou extrahovány jednotlivé snímky. V dalším kroku jsou snímky převedeny do barevné reprezentace BGR. Následně je provedeno předzpracování snímku a upravena jeho velikost v závislosti na použitých vstupních parametrech. V dalším kroku jsou získány jednotlivé predikce na základě před-trénovaného modelu a odečten čas potřebný pro výpočet této predikce. Po úspěšném získání predikčních oblastí je upraveno měřítko daného snímku a extrahovány ty predikce, které jsou vetší než stanovený práh. Ve výchozím stavu je práh nastaven na hodnotu 0.5, ale i tuto hodnotu lze při spuštění aplikace nastavit příslušným parametrem.

Konečným krokem je zakreslení hraničních oblastí nalezených objektů do konkrétního snímku a jeho zobrazení uživateli. Ve výchozím stavu je počet vykreslených hraničních oblastí omezen na jedenu, avšak i tato hodnota je nastavitelná použitím příslušného parametru při spuštění aplikace. V případě, že byla při spuštění aplikace zvolena možnost pro ukládání

snímků, jsou všechny zpracované snímky uloženy na disk pro potřeby dalšího zpracování. Algoritmus pro zpracování snímku a nalezení objektu je stručně popsán zde:

Algoritmus 1: Pseudokód aplikace k detekci zbraní

Vstup:
model: natrénovaný model ve formátu h5
zdroj: zdroj obrazových dat
threshold: hraniční hodnota úspěšné detekce

```
Zpracování a validace vstupních parametrů
Načtení natrénovaného modelu
Načtení snímků
for obrázek in zdroj do
    Převod obrázku do BGR
    Předzpracování snímků a případná změna velikosti
    Získání predikcí na základě před-trénovaného modelu
    Seřazení predikcí a získání maximálního počtu predikcí
    for predikce in predikcích do
        if skóre predikce > treshold then
            Zakreslení ohraničení
            Zakreslení třídy a celkového skóre
        end
    end
    Náhled snímku
end
Uvolnění zdrojů
Ukončení aplikace
```

Kapitola 5

Výsledky experimentů

Vyhodnocení výsledků natrénovaného modelu bylo vyhodnoceno pomocí aplikace **evaluate**. Aplikace pro vyhodnocení přesnosti daného modelu využívá vyhodnocovací metriku nazývanou *mean Average Precision*(mAP). Tato vyhodnocovací metrika je využívaná v oblastech zabývajících se detekcí objektů. Výpočet hodnoty mAP je složen ze dvou složek, a to ze složky *precission* viz rovnice 5.1, která udává procento správných předpovědí a složky *recall* viz rovnice 5.2, jenž udává správnost nalezení všech pozitivních výsledků.

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{5.1}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{5.2}$$

K získání skutečně pozitivních výsledků je zde využita další metrika nazývaná *Intersection over union*(IoU). Tuto metriku lze využít na jakýkoliv algoritmus, jehož výstupem jsou predikce ohraničujících boxů. Pro tuto metriku je nutné znát přesnou polohu skutečného ohraničujícího boxu daného objektu a polohu predikovaného boxu. Metrika IoU udává míru překrytí mezi dvěma ohraničujícími boxy. Používá se tedy k měření toho, do jaké míry se predikovaná oblast překrývá se skutečnou hranicí objektu. Výsledná hodnota je porovnávaná s prahovou hodnotou (nejčastěji 0,5). Pokud je výsledná hodnota větší než prahová hodnota, je detekce označena jako skutečně pozitivní, v opačném případě je hodnota označena za falešně pozitivní.

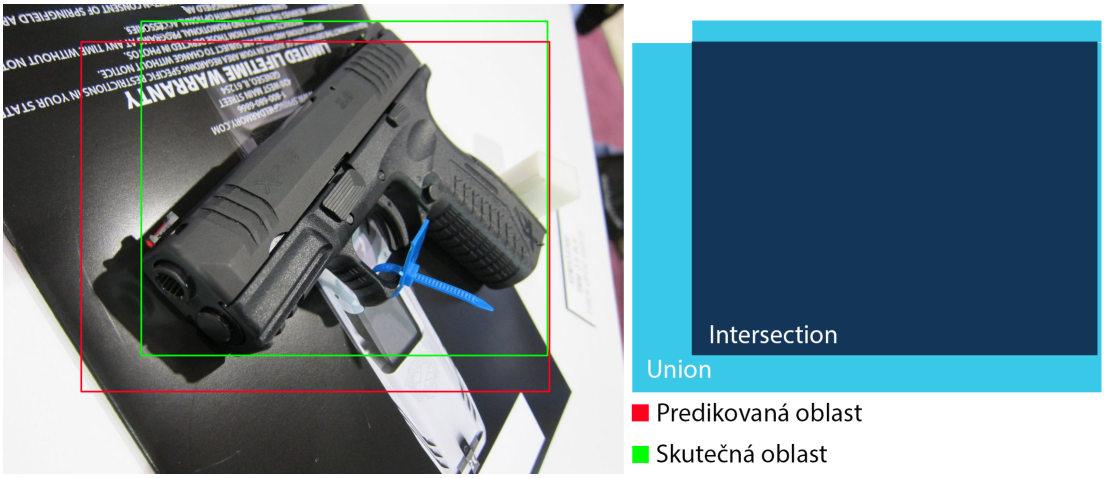
Výpočet hodnoty IoU je definován v rovnici 5.3 a na obrázku 5.1 jsou znázorněny jednotlivé oblasti na reálném případě.

$$IoU = \frac{Intersection}{Union} \tag{5.3}$$

5.1 Úspěšnost natrénovaných modelů

Jak již bylo popsáno v kapitole 4.4.3 byly natrénovány celkem 3 modely v závislosti na použitém datovém souboru a počtu iterací při trénování. K vyhodnocení úspěšnosti jednotlivých modelů byly vytvořeny dvě testovací skupiny. Na základě těchto tří skupin byla vyhodnocena úspěšnost natrénovaného modelu na základě různých testovacích dat.

Všechny snímky, na nichž je zachycen typ zbraně využitý v procesu generování, obsahují na snímku pouze jednu nezakrytou zbraň. Výjimku tvoří třída krátkých zbraní, která



Obrázek 5.1: Obrázek znázorňuje jednotlivé plochy využívané k výpočtu hodnoty IoU. Na pravém obrázku jsou znázorněny ohraničující oblasti, kde červené ohraničení označuje predikovanou oblast modelem a zelené ohraničení označuje skutečnou oblast, ve které se sledovaný objekt ve skutečnosti nachází. Na levém obrázku jsou tyto oblasti znázorněny v podobě ploch využívaných pro výpočet IoU.

obsahuje i snímky částečně zakrytých zbraní. Pro vyhodnocení úspěšnosti modelu byla využita aplikace **evaluate**, jenž je součástí knihovny Keras RetinaNet. Jelikož snímky zbraní obsahují vždy jen jednu zbraň, byla aplikace nastavena na detekci pouze jedné zbraně. Výsledky trénování modelů v různých fázích trénování a různých typech datových souborů jsou znázorněny níže.

První skupinu testovacích dat tvoří 3 173 snímků zbraní, které byly odděleny od sady snímků určených k trénování modelu. Tyto snímky jsou tak tvořeny obrázky zbraní pořízených při procesu generování a zachycují 3D model zbraně na různém pozadí. Snímky tedy neobsahují reálné zbraně. Výsledné hodnoty testování modelu na této testovací sadě je znázorněno v tabulce 5.1 a sloužilo výhradně pro vyhodnocování úspěšnosti modelu během trénovací fáze. Z tabulky je patrné, že v prvotních epochách je již úspěšnost velmi vysoká. Tento jev může být způsoben tím, že pro trénování bylo využito velké množství obrázků, ale základní model zůstával stejný a měnilo se pouze osvětlení a orientace modelu. Rovněž testovací data obsahují snímky modelů zbraní použitých pro trénování. Díky tomu se model mohl velmi rychle naučit rozpoznávat konkrétní model zbraně.

Epocha	Krátké zbraně	Dlouhé zbraně	Automatické zbraně	mAP
10	0,958	0,8372	0,9772	0,9242
20	0,9337	0,743	0,9718	0,8828
30	0,9465	0,7883	0,9762	0,9037
40	0,9536	0,7489	0,9761	0,8928
50	0,9381	0,7696	0,9762	0,8946

Tabulka 5.1: Znázorňuje úspěšnost detekcí jednotlivých tříd a celkové úspěšnosti modelu v rámci jednotlivých epoch. Testování modelu probíhalo na testovací sadě obsahující 3 173 snímků modelů zbraní.

Druhou testovací skupinu dat tvoří 520 obrázků reálných zbraní získaných z různých zdrojů. Testovací skupina obsahuje 278 snímků krátkých zbraní, 83 snímků dlouhých zbraní

a 163 snímků automatických zbraní. Výběr snímků reálných zbraní probíhal s ohledem na jednotlivé modely použitých během procesu generování snímků. Jednotlivé třídy obsahují rozdílné počty zbraní a rozdílné počty typů. Například třída automatických zbraní obsahuje snímky tří modelů zbraní, konkrétně Avtomat Kalašnikova obrazca 47 (AK-47), Heckler & Koch MP5 a Heckler & Koch HK416, oproti tomu třída dlouhých zbraní obsahuje pouze model brokovnice a pušky Winchester. Jelikož bylo žádoucí získat reálnou představu o celkové úspěšnosti detekce reálných zbraní na základě modelu natrénovaného na snímcích 3D modelů zbraní, obsahují testovací snímky pouze typy reálných zbraní použitých při procesu generování.

Výsledky testování na reálných zbraních na modelu, který v rámci jedné epochy vykonával 10 000 operací, je znázorněn v tabulce 5.2. Z výsledků je patrné, že model nejlépe detekuje krátké zbraně. Tato skutečnost je způsobena použitím 2 466 reálných krátkých zbraní. Ostatní třídy jsou pro detekci zbraní takřka nepoužitelné. Velmi podobné výsledky zobrazené v tabulce 5.3 vykazuje i model trénovaný na stejných datech pouze s rozdílným počtem iterací. Tato skutečnost je nejspíše způsobena malým počtem modelů při generování zbraně. Těchto modelů by muselo být mnohonásobně více a to i pro jeden konkrétní typ, aby pokryl širokou škálu reálných zbraní.

Epocha	**Krátké zbraně**	**Dlouhé zbraně**	**Automatické zbraně**	**mAP**
10	0,6878	0,0499	0,1741	0,3039
20	0,6502	0,0161	0,0563	0,2408
30	0,6396	0	0,0563	0,2319
40	0,6266	0	0,025	0,2172
50	0,6692	0,0241	0,0563	0,2498

Tabulka 5.2: Znázorňuje úspěšnost detekcí jednotlivých tříd a celkové úspěšnosti modelu v rámci jednotlivých epoch při 10 000 iteracích. Testování modelu probíhalo na testovací sadě obsahující 520 obrázků reálných zbraní.

Epocha	**Krátké zbraně**	**Dlouhé zbraně**	**Automatické zbraně**	**mAP**
10	0,6723	0	0,075	0,2491
20	0,6934	0,0241	0,05	0,2558
30	0,6926	0,0241	0,0688	0,2618
40	0,698	0,0241	0,0688	0,2636
50	0,698	0,0241	0,0688	0,2636

Tabulka 5.3: Znázorňuje úspěšnost detekcí jednotlivých tříd a celkové úspěšnosti modelu v rámci jednotlivých epoch při 5 000 iteracích. Testování modelu probíhalo na testovací sadě obsahující 520 obrázků reálných zbraní.

Posledním testovaným modelem byl model trénovaný pouze na modelových zbraních. Trénovací data neobsahovala žádnou reálnou zbraň. Tento model byl trénován pouze pro testovací účely. Cílem bylo získat představu o tom, jak bude model úspěšný při detekci reálných zbraní, pokud bylo pro trénování využito pouze modelových dat. Z tabulky 5.4 je patrné, že přesnost jednotlivých tříd se pohybuje v rámci jednotek procent. Pokud by mělo být detekování úspěšné muselo by být vygenerováno několik stovek tisíc obrázků na základě stovek různých modelů.

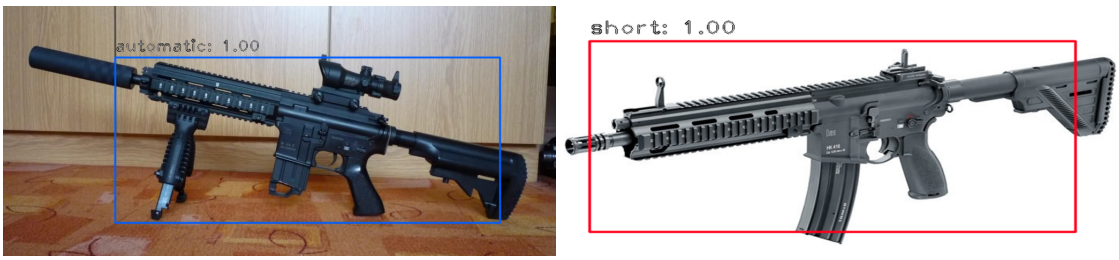
Epocha	Krátké zbraně	Dlouhé zbraně	Automatické zbraně	mAP
10	0,0051	0,0094	0,0425	0,019
20	0,0054	0,0281	0,0323	0,0219
30	0,0008	0,0249	0,0467	0,0241

Tabulka 5.4: Znázorňuje úspěšnost detekcí jednotlivých tříd a celkové úspěšnosti modelu trénovaném pouze na modelových datech. Testování modelu probíhalo na testovací sadě obsahující 520 obrázků reálných zbraní.

5.2 Detekce zbraní na konkrétních ukázkách

K detekci zbraní byl použitý natrénovaný model po 50 epochách, kdy v rámci jedné epochy probíhalo 50 iterací. Při testování na přiložených videích vykazoval tento model nejlepší výsledky. Celkové výsledky testování jednotlivých modelů byly popsány v předchozí kapitole, zde budou zobrazeny a stručně okomentovány konkrétní snímky. Testování probíhalo jak na připraveném testovacím souboru obsahujících 520 snímků reálných zbraní, tak na několika krátkých videích získaných z různých zdrojů internetu. Jak obrázky, tak videa byla vybírána s ohledem na použitý typ zbraní při generování datového souboru. Jelikož byla variace použitých modelů velmi malá, postrádalo zde testování na novém typu zbraně smysl.

První obrázek 5.2 zobrazuje úspěšnou a neúspěšnou detekci automatické zbraně Heckler & Koch HK416. Ačkoliv se jedná o snímky, na kterých je zbraň dobře viditelná a podobná modelu zbraně použité při generování, byla detekce úspěšná jen částečně. Na obou obrázcích byla zbraň úspěšně detekována, avšak na pravém obrázku je zbraň zařazena do nesprávné kategorie. Na testovacích snímcích tohoto typu zbraně byla zbraň úspěšně detekována na všech snímcích, avšak na drtivé většině snímků byla zbraň zařazena do kategorie krátkých zbraní.



Obrázek 5.2: Obrázek vlevo zobrazuje úspěšnou detekci automatické zbraně Heckler & Koch HK416. Na obrázku vpravo byla zbraň úspěšně detekována, avšak začleněna do nesprávné kategorie.

Druhý obrázek 5.3 znázorňuje úspěšnou a neúspěšnou detekci pušky Winchester z kategorie dlouhých zbraní. Tato kategorie vykazuje nejhorší výsledky ze všech kategorií. Na vině bude s největší pravděpodobností použití pouze dvou typů velmi rozdílných modelů zbraně, což k možné variaci reálných zbraních tohoto typu je rozpoznávací schopnost detektoru velmi omezena. Z celkového počtu 50 reálných snímků tohoto typu zbraně byl úspěšně rozpoznán a správně kategorizován pouze 1 snímek. Zbraň nebyla vůbec detekována na 7 snímcích, a ve zbylých případech byla úspěšně detekována, ale nevhodně kategorizována do kategorie krátkých zbraní.

Poslední kategorií byla kategorie krátkých zbraní. Na obrázku 5.4 jsou znázorněny snímky úspěšné detekce a kategorizace zbraní na jednoduchých snímcích. Zde si detektor

Obrázek 5.3: Horní obrázek znázorňuje úspěšnou detekci i správně zařazenou pušku Winchester. Obrázek vlevo je zobrazena úspěšně detekovaná zbraň, ale nesprávně zařazena do kategorie krátkých zbraní. Na obrázku vpravo nebyl detektor schopen zbraň detekovat.

vedl velmi dobře. Dokázal detekovat a správně kategorizovat převážnou většinu testovacích zbraní.

Obrázek 5.4: Na obrázku je znázorněna úspěšná detekce krátkých zbraní.

Z důvodů použití reálných, částečné zakrytých zbraní při trénování modelu v této kategorii byla zde zkoumána detekce na poměrně komplikovaných obrázcích, jak je znázorněno na obrázku 5.5. Zde si však model vedl velmi dobře a dokázal rozpoznat a správně kategorizovat krátkou zbraň.

I když v rámci této diplomové práce mělo probíhat testování pouze na obrázcích zbraní, bylo zde provedeno optické vyhodnocení detekce zbraní i na videích obsahujících zbraň. V první fázi byl detektor testován na videích zachycující modely zbraní podobným těm, které sloužily pro generování trénovacích snímků, avšak nejedná se o ty stejné modely.

Obrázek 5.5: Na obrázku jsou znázorněny poměrně složité snímky s úspěšnou detekcí krátkých zbraní.

Zde byl model velmi přesný, což je dle mého názoru způsobeno zmíněnou podobností modelů a celkovou jednoduchostí scény, která neobsahuje žádné další objekty a rušivé elementy.

V další fázi bylo testování detektoru na videích obsahující reálné zbraně ve většině případů i značně zakryté. Zde byl schopen detektor poměrně přesně detekovat zbraň, avšak kategorizování zbraně bylo velmi nepřesné. V drtivé většině případů byla detekovaná zbraň zařazena do kategorie krátkých zbraní. Což je způsobeno tím, že pouze kategorie krátkých zbraní byla při trénovací fázi obohacena o variaci snímků krátkých zbraní včetně těch částečně zakrytých.

Porovnání výsledků z již existujícími řešeními není zcela objektivní, jelikož zde byla zkoumána možnost detekce zbraní při trénování modelu na snímcích pořízených na základě modelu zbraně. Z tohoto pohledu je celková úspěšnost modelu velmi nízká, avšak tento výsledek je kombinací několika zásadních faktorů. Prvním z nich je použití velmi malého množství různých 3D modelů zbraní, čímž je velmi těžké pokrýt variaci reálných zbraní. Další neméně důležitou roli, zde hraje kvalita použitých modelů. Pro tuto práci byly až na výjimku vybrány modely zdarma a tudíž tomu odpovídala výsledná kvalita modelu, čímž došlo k jejich další redukci. Dále je důležité si uvědomit, že na snímcích reálných zbraní se vyskytují i další faktory ovlivňující výslednou podobu zbraně.

Při zkoumání již existujících řešení nebyla nalezena práce, která by jednak zbraň detekovala a následně ještě rozčleňovala do kategorie krátkých, dlouhých a automatických zbraní. Avšak při celkovém testování modelu napříč všemi kategoriemi dominovala v úspěšnosti kategorie krátkých zbraní. I při testování na kategoriích dlouhých a automatických zbraní byla ve většině případu zbraň detekována, avšak nevhodně kategorizována nejčastěji do kategorie krátkých zbraní. Z tohoto pohledu by mohla být úspěšnost této kategorie prohlášena za celkovou úspěšnost modelu pro detekci zbraní. V porovnání s metodou 2.2.3, kde se přesnost pohybovala kolem 85 % a metodou 2.2.3 s přesností 89 % dosahuje natrénovaný model s úspěšností přibližně 67 % horších výsledků. Však toto zhodnocení není z výše zmíněných důvodů zcela objektivní.

Kapitola 6

Závěr

V této diplomové práci byla zkoumána možnost detekce zbraní ve scéně a implementace aplikace, která bude schopná danou zbraň ve scéně najít a určit, zda se jedná o krátkou zbraň, dlouhou zbraň nebo zbraň automatickou. Určit typ zbraně může v reálném nasazení aplikace pomoci při rozhodování, jakou bezpečnostní složku k incidentu vyslat. Dnešní doba nabízí spoustu možností, jak k dané problematice přistupovat. Jednou z možností, která je dnes na obrovském vzestupu a dotýká se mnoha odvětvích, mezi které patří, zdravotnictví, strojírenství a další, bezpochyby patří strojové učení. Tato práce pro řešení problému využívá metodu hloubkového učení. Základním předpokladem strojového učení je dostatečná trénovací sada, na které bude model trénován. Tato trénovací sada musí být navíc předem anotována, aby byla neuronová síť schopna určit objekt zájmu.

Nejdříve bylo nutné získat dostatečně velkou datovou sadu, na které by následně proběhl proces trénování. Jelikož je pro fázi trénování nutné znát nejen přesnou polohu zbraně ve scéně, ale i třídu, do které zbraň spadá, bylo nutné jednotlivé snímky přesně anotovat. Tento proces je však nesmírně časově náročný. Proto bylo rozhodnuto o pokusu generování snímků z existujících 3D modelů zbraní. Pro tyto účely byla vyvinuta webová aplikace umožnující generování snímků zbraní na základě různých modelů. Tato aplikace umožňuje nejen manipulaci s modelem, ale i výběr pozadí a osvětlení. Další nepostradatelnou vlastností aplikace je možnost získání velmi přesné polohy zbraně ve snímku s přesností na 1 pixel. Díky tomu je možné celý proces anotovaní snímků provést strojově. Tímto způsobem bylo vygenerováno 24 226 snímků zbraní za pomoci 7 různých modelů zbraní. Pro třídu krátkých zbraní bylo navíc použito 2 466 snímků reálných zbraní.

Další fází bylo natrénování modelu na vytvořeném datovém souboru. Pro účely trénování byla zvolena již existující implementace modelu RetinaNet. Výsledkem bylo natrénování celkem tří modelů na různých datových sadách. Pro demonstraci fungování natrénovaného modelu byla vyvinuta aplikace umožnující detekci zbraně v obrázku nebo videozáznamu. Celkové vyhodnocení výsledků probíhalo na dvou testovacích sadách. První z nich obsahovala pouze snímky modelových zbraní. Zde byl model velmi přesný, což bylo způsobeno použitím stejného modelu zbraně pro trénovací snímky i pro testovací snímky. Druhá testovací sada byla tvořena 520 snímky reálných zbraní. Zde byl model o poznání horší. Jedinou použitelnou třídou byla třída krátkých zbraní, která při trénování využívala i reálné zbraně. Avšak ostatní dvě třídy vykazovaly úspěšnost v řádu jednotek procent. Tato skutečnost je způsobena tím, že reálné zbraně, ať už jsou stejného typu, mohou být v různých barvách a provedeních, což s takto malým počtem modelových zbraní není možné pokrýt. Jednotlivé vygenerované snímky obsahují vždy jen jednu nezakrytou zbraň.

Další možnosti vývoje jsou následující. Rozšířit aplikaci pro generování snímků zbraní o možnost generování více zbraní zároveň. V případě více zbraní ve scéně a použitím vhodného osvětlení by mohl ve scéně vzniknout nový parametr v podobě stínu. Tímto by se generované snímky více přiblížily skutečným snímkům. Další důležitou vlastností, vyskytujících se u reálných snímků je, že zbraň je na snímku z části někdy i velké části zakrytá. Proto by bylo vhodné upravit existující modely tak, aby byly částečně zakryty například modelem ruky. Dále by bylo vhodné získat větší počet kvalitních modelů zbraní, aby byla pokryta variace reálných zbraní.

Literatura

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

[2] Abidi, B.; Zheng, Y.; Gribok, A.; aj.: Improving Weapon Detection in Single Energy X-Ray Images Through Pseudocoloring. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, ročník 36, 12 2006: s. 784 – 796, doi:10.1109/TSMCC.2005.855523.

[3] Alahi, A.; Ortiz, R.; Vandergheynst, P.: FREAK: Fast retina keypoint. 06 2012, s. 510–517, doi:10.1109/CVPR.2012.6247715.

[4] Aurelien, G.: *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. OReilly Media, 2017.

[5] Baştan, M.; Yousefi, M. R.; Breuel, T. M.: Visual Words on Baggage X-Ray Images. In *Computer Analysis of Images and Patterns*, editace P. Real; D. Diaz-Pernil; H. Molina-Abril; A. Berciano; W. Kropatsch, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, ISBN 978-3-642-23672-3, s. 360–368.

[6] Ben Halima, N.; Hosam, O.: Bag of Words Based Surveillance System Using Support Vector Machines. *International Journal of Security and Its Applications*, ročník 10, 04 2016: s. 331–346, doi:10.14257/ijsia.2016.10.4.30.

[7] Bolfing, A.; Halbherr, T.; Schwaninger, A.: How image based factors and human factors contribute to threat detection performance in X-ray aviation security screening. 11 2008, s. 419–438, doi:10.1007/978-3-540-89350-9_30.

[8] Chollet, F.: *Deep Learning with Python*. USA: Manning Publications Co., první vydání, 2017, ISBN 1617294438.

[9] Chollet, F.: About Keras. online, 2020.
URL https://keras.io/about/

[10] Dalal, N.; Triggs, B.: Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, ročník 1, June 2005, ISSN 1063-6919, s. 886–893 vol. 1, doi:10.1109/CVPR.2005.177.

[11] Dargan, S.; Kumar, M.; Maruthi; aj.: A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning. 07 2019.

[12] Dertat, A.: AI Starter- Build your first Convolution neural network in Keras from scratch to perform. online, 2017.
URL https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

[13] Desarda, A.: Understanding AdaBoost. Jan 2019.
URL https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

[14] Everingham, M.; Eslami, S. M. A.; Van Gool, L.; aj.: The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, ročník 111, č. 1, Leden 2015: s. 98–136.

[15] Felzenszwalb, P. F.; Girshick, R. B.; McAllester, D. A.; aj.: Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 32, 2009: s. 1627–1645.

[16] Flitton, G.; Breckon, T.; Megherbi, N.: A comparison of 3D interest point descriptors with application to airport baggage object detection in complex CT imagery. *Pattern Recognition*, ročník 46, 09 2013: str. 2420–2436, doi:10.1016/j.patcog.2013.02.008.

[17] Foundation, P. S.: What is Python? Executive Summarye. online, 2020.
URL https://www.python.org/doc/essays/blurb/

[18] Gaiser, H.; de Vries, M.; Lacatusu, V.; aj.: fizyr/keras-retinanet 0.5.1. Červen 2019, doi:10.5281/zenodo.3250670.
URL https://doi.org/10.5281/zenodo.3250670

[19] Girshick, R.: Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, s. 1440–1448.

[20] Girshick, R. B.; Donahue, J.; Darrell, T.; aj.: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2013: s. 580–587.

[21] Guo, Y.; Liu, Y.; Oerlemans, A.; aj.: Deep learning for visual understanding: A review. *Neurocomputing*, ročník 187, 11 2015, doi:10.1016/j.neucom.2015.09.116.

[22] Harris, C.; Stephens, M.: A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, 1988, s. 147–151.

[23] He, K.; Gkioxari, G.; Dollár, P.; aj.: Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, ISSN 2380-7504, s. 2980–2988, doi:10.1109/ICCV.2017.322.

[24] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, s. 770–778.

[25] Hochreiter, S.: The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, ročník 6, č. 2, Duben 1998: str. 107–116, ISSN 0218-4885, doi:10.1142/S0218488598000094.
URL https://doi.org/10.1142/S0218488598000094

[26] Howard, A. G.; Zhu, M.; Chen, B.; aj.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv*, ročník abs/1704.04861, 2017.

[27] Karpathy, A.: What I learned from competing against a ConvNet on ImageNet. online, 2014.
URL http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

[28] Khongpit, V.; phangphol, U.; Kasemsan, K.; aj.: 2D Gun's type classification using edge detection algorithm and SUSAN low level image processing. In *2012 International Conference for Internet Technology and Secured Transactions*, Dec 2012, ISSN null, s. 521–523.

[29] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, editace F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, s. 1097–1105.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[30] Kuznetsova, A.; Rom, H.; Alldrin, N.; aj.: The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *CoRR*, ročník abs/1811.00982, 2018, 1811.00982.
URL http://arxiv.org/abs/1811.00982

[31] Lecun, Y.; Bottou, L.; Bengio, Y.; aj.: Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, ročník 86, 12 1998: s. 2278 – 2324, doi:10.1109/5.726791.

[32] Lin, T.; Dollár, P.; Girshick, R. B.; aj.: Feature Pyramid Networks for Object Detection. *CoRR*, ročník abs/1612.03144, 2016, 1612.03144.
URL http://arxiv.org/abs/1612.03144

[33] Lin, T.; Goyal, P.; Girshick, R.; aj.: Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 42, č. 2, Feb 2020: s. 318–327, ISSN 1939-3539, doi:10.1109/TPAMI.2018.2858826.

[34] Lin, T.-Y.; Maire, M.; Belongie, S.; aj.: Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, editace D. Fleet; T. Pajdla; B. Schiele; T. Tuytelaars, Cham: Springer International Publishing, 2014, s. 740–755.

[35] Liu, L.; Ouyang, W.; Wang, X.; aj.: Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision*, Oct 2019, ISSN 1573-1405, doi:10.1007/s11263-019-01247-4.
URL https://doi.org/10.1007/s11263-019-01247-4

[36] Liu, W.; Anguelov, D.; Erhan, D.; aj.: SSD: Single Shot MultiBox Detector. 2016, to appear.
URL http://arxiv.org/abs/1512.02325

[37] Lopez, A. D.; Kollialil, E. S.; Gopan, K. G.: Adaptive Neuro-fuzzy Classifier for Weapon Detection in X-Ray Images of Luggage Using Zernike Moments and Shape

Context Descriptor. In *2013 Third International Conference on Advances in Computing and Communications*, Aug 2013, ISSN null, s. 46–49, doi:10.1109/ICACC.2013.16.

[38] Lowe, D. G.: Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, ročník 2, Sep. 1999, ISSN null, s. 1150–1157 vol.2, doi:10.1109/ICCV.1999.790410.

[39] Ma, X.; Li, B.; Zhang, Y.; aj.: The Canny Edge Detection and Its Improvement. In *Artificial Intelligence and Computational Intelligence*, editace J. Lei; F. L. Wang; H. Deng; D. Miao, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33478-8, s. 50–58.

[40] Microsoft: Why did we build Visual Studio Code. online, 2020.
URL https://code.visualstudio.com/docs/editor/whyvscode

[41] Nercessian, S.; Panetta, K.; Agaian, S.: Automatic Detection of Potential Threat Objects in X-ray Luggage Scan Images. In *2008 IEEE Conference on Technologies for Homeland Security*, May 2008, ISSN null, s. 504–509, doi:10.1109/THS.2008.4534504.

[42] NVIDIA: NVIDIA cuDNN. online, 2020.
URL https://developer.nvidia.com/cudnn

[43] Oertel, C.; Bock, P.: Identification of Objects-of-Interest in X-Ray Images. In *35th IEEE Applied Imagery and Pattern Recognition Workshop (AIPR'06)*, Oct 2006, ISSN 2332-5615, s. 17–17, doi:10.1109/AIPR.2006.25.

[44] Olmos, R.; Tabik, S.; Herrera, F.: Automatic Handgun Detection Alarm in Videos Using Deep Learning. *Neurocomput.*, ročník 275, č. C, Leden 2018: s. 66–72, ISSN 0925-2312.
URL http://dl.acm.org/citation.cfm?id=3198485.3198798

[45] OpenCV: About. online, 2020.
URL https://opencv.org/about/

[46] Osuna, E.; Freund, R.; Girosit, F.: Training support vector machines: an application to face detection. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 1997, ISSN 1063-6919, s. 130–136, doi:10.1109/CVPR.1997.609310.

[47] Pallawi: AI Starter- Build your first Convolution neural network in Keras from scratch to perform. online, 2019.
URL https://mc.ai/ai-starter-build-your-first-convolution-neural-network-in-keras-from-scratch-to-perform/

[48] Rashid, T.: *Make Your Own Neural Network Tariq Rashid*. CreateSpace Independent Publishing Platform, 2016.

[49] Redmon, J.; Divvala, S.; Girshick, R.; aj.: You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, s. 779–788.

[50] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *CoRR*, ročník abs/1612.08242, 2016, 1612.08242.
URL http://arxiv.org/abs/1612.08242

[51] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *CoRR*, ročník abs/1804.02767, 2018, 1804.02767.
URL http://arxiv.org/abs/1804.02767

[52] Ren, S.; He, K.; Girshick, R.; aj.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 01 2016, s. 1–10.

[53] Rowley, H. A.; Baluja, S.; Kanade, T.: Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 20, č. 1, Jan 1998: s. 23–38, ISSN 1939-3539, doi:10.1109/34.655647.

[54] Schmid, C.; Mohr, R.: Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 19, č. 5, May 1997: s. 530–535, ISSN 1939-3539, doi:10.1109/34.589215.

[55] Sermanet, P.; LeCun, Y.: Traffic Sign Recognition with Multi-Scale Convolutional Networks. In *Proceedings of International Joint Conference on Neural Networks (IJCNN'11)*, 2011, s. 2809–2813.

[56] Shrivastava, A.; Mulam, H.; Girshick, R.: Training Region-Based Object Detectors with Online Hard Example Mining. 06 2016, s. 761–769, doi:10.1109/CVPR.2016.89.

[57] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, ročník abs/1409.1556, 2014.
URL http://arxiv.org/abs/1409.1556

[58] Singleton, M.; Taylor, B.; Taylor, J.; aj.: Gun Identification Using Tensorflow. 10 2018, doi:10.1007/978-3-030-00557-3_1.

[59] Smith, S.: SUSAN.
URL https://users.fmrib.ox.ac.uk/~steve/susan/susan/susan.html

[60] Swanson, J.; Sampson, N.; Petukhova, M.; aj.: Guns, Impulsive Angry Behavior, and Mental Disorders: Results from the National Comorbidity Survey Replication (NCS-R). *Behavioral sciences & the law*, ročník 33, 04 2015, doi:10.1002/bsl.2172.

[61] Szegedy, C.; Wei Liu; Yangqing Jia; aj.: Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, s. 1–9.

[62] TradeSmart.cz: Co je overfitting? Trading Terminologie! online, 2018.
URL https://www.tradesmart.cz/co-je-overfitting-trading-terminologie/

[63] Uijlings, J.; Sande, K.; Gevers, T.; aj.: Selective Search for Object Recognition. *International Journal of Computer Vision*, ročník 104, 09 2013: s. 154–171, doi:10.1007/s11263-013-0620-5.

[64] Verma, G.; Tiwari, R.: A Computer Vision based Framework for Visual Gun Detection Using Harris Interest Point Detector. 08 2015, doi:10.1016/j.procs.2015.06.083.

[65] Viola, P.; Jones, M. J.: Robust Real-Time Face Detection. *International Journal of Computer Vision*, ročník 57, č. 2, May 2004: s. 137–154, ISSN 1573-1405, doi:10.1023/B:VISI.0000013087.49260.fb.
URL https://doi.org/10.1023/B:VISI.0000013087.49260.fb

[66] Zeiler, M. D.; Fergus, R.: Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, editace D. Fleet; T. Pajdla; B. Schiele; T. Tuytelaars, Cham: Springer International Publishing, 2014, ISBN 978-3-319-10590-1, s. 818–833.

[67] Zhang, X.; Yang, Y.-H.; Han, Z.; aj.: Object Class Detection: A Survey. *ACM Comput. Surv.*, ročník 46, č. 1, Červenec 2013, ISSN 0360-0300, doi:10.1145/2522968.2522978.
URL https://doi.org/10.1145/2522968.2522978

[68] Zhu, P.; Wen, L.; Bian, X.; aj.: Vision Meets Drones: A Challenge. *CoRR*, ročník abs/1804.07437, 2018, 1804.07437.
URL http://arxiv.org/abs/1804.07437

[69] Zou, Z.; Shi, Z.; Guo, Y.; aj.: Object Detection in 20 Years: A Survey. *CoRR*, ročník abs/1905.05055, 2019, 1905.05055.
URL http://arxiv.org/abs/1905.05055