VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

DETEKCE A KLASIFIKACE LÉTAJÍCÍCH OBJEKTŮ
DETECTION AND CLASSIFICATION OF FLYING OBJECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Tomáš Jurečka**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ilona Janáková, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Tomáš Jurečka ***ID:*** 197714

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Detekce a klasifikace létajících objektů

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je navrhnout systém počítačového vidění pro detekci, trasování a klasifikaci létajících objektů na obloze. Pro klasifikaci se předpokládá využití konvolučních neuronových sítí a klasifikace do kategorií typu: letadlo, dron, pták, hmyz, případně další.

1. Seznamte se s danou problematikou. Proveďte rešerši existujících přístupů.
2. Navrhněte vhodnou kombinaci a uspořádání hardwarových komponent – kamera s optikou + PC/minipočítač.
3. Pořiďte dostatečně rozsáhlou a pestrou databázi reálných snímků/sekvencí.
4. Navrhněte a realizujte algoritmy detekce a trasování objektů.
5. Navrhněte a realizujte klasifikátor objektů.
6. Vše otestujte. Definujte omezující podmínky. Zhodnoťte.

DOPORUČENÁ LITERATURA:

A. Schumann, L. Sommer, J. Klatte, T. Schuchert and J. Beyerer, "Deep cross-domain flying object classification for robust UAV detection," 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Lecce, 2017, pp. 1-6, doi: 10.1109/AVSS.2017.8078558.

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 17.5.2021

Vedoucí práce: Ing. Ilona Janáková, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

Abstrakt

Práce se zabývá detekcí a klasifikací létajících objektů. Postup lze rozdělit na tři části. V první části je tvořen dataset létajících objektů. Pro tvorbu datasetu je použita metoda reverzního vyhledávání obrázků. Další část tvoří rešerše algoritmů pro detekci, trasování a klasifikaci. Následně jsou jednotlivé algoritmy aplikovány a zhodnoceny. V poslední části je proveden návrh hardwarových komponent.

Klíčová slova

Dataset, reverzní vyhledání obrázků, detekce, trasování, klasifikace, YOLO

Abstract

The thesis deals with the detection and classification of flying objects. The work can be divided into three parts. The first part describes the creation of dataset of flying objects. The reverse image search is used to create the dataset. The next part is a research of algorithms for detection, tracking and classification. Subsequently, the individual algorithms are applied and evaluated. In the last part, the design of hardware components is performed.

Keywords

Dataset, reverse image search, detection, tracking, classification, YOLO

Bibliografická citace

JUREČKA, Tomáš. Detekce a klasifikace létajících objektů [online]. Brno, 2021 [cit. 2021-05-14]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/134525. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Ilona Janáková.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Tomáš Jurečka*
VUT ID studenta:	*197714*
Typ práce:	*Diplomová práce*
Akademický rok:	*2020/21*
Téma závěrečné práce:	*Detekce a klasifikace létajících objektů*

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

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 17. května 2021 ----------

podpis autora

Poděkování

Děkuji vedoucí diplomové práce Ing. Iloně Janákové, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: 17. května 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Člověk je už odpradávna fascinován schopností ptáků létat. Z Řecké mytologie známe příběh o Daidalovi, který sestrojil dva páry křídel. První písemná zmínka o letu člověka se datuje na začátek jedenáctého století. Mnich Eilmer z Malmesbury zdolal pomocí kluzáku vzdálenost přes dvě stě metrů. Další historický úspěch o podmanění oblohy byl zaznamenán v roce 1783, kdy se podařilo bratrům Montgolfierům vyslat do oblak horkovzdušný balón. Posádku tvořila ovce, kachna a kohout. Následně ve stejném roce absolvovala let i lidská posádka Jean-François Pilâtre de Rozier a François Laurent d'Arlandes. První motorové letadlo sestrojili bratři Wrightové. První let se uskutečnil v roce 1903. Od této doby jsme svědky konstantní evoluce v letectví. V překvapivě krátké době se z létání stala poměrně běžná záležitost.

V nedávné době byl v soukromém sektoru zaznamenán rozmach dálkově řízených letounů. Nejpopulárnější provedení je quadrokoptéra s vestavěnou kamerou. Tyto letouny jsou nejčastěji pořizovány za účelem natáčení videí a zábavy při pilotování.

Při provozu dálkově řízených letounů můžou vznikat nežádoucí situace. Dálkově řízené letouny mohou ohrožovat jiné létající prostředky, a to včetně těch s lidskou posádkou. Pomocí vestavěné kamery může být narušeno soukromí. V neposlední řadě je hrozbou militantní modifikace letounu.

V současné době jsou již vyvinuty prostředky pro zneškodnění těchto letounů. Elegantním způsobem je přebrání kontroly nad letounem. Poté stačí například nařídit letounu vrácení na místo vzletu. Účinné je rovněž použití mechanických prostředků. Například vystřelení sítě.

Cílem práce je navrhnout jednoduché řešení pro detekci a klasifikaci létajících objektů.

Nejprve bude proveden teoretický úvod zaměřený na konvoluční neuronové sítě. Jejich použití je v současné době velmi populární a výkon modelů rok od roku roste. Budou popsány klasifikační modely neuronových sítí, které na základě vstupního obrázku určují jeho třídu. Dalším typem neuronových sítí jsou modely, které jsou schopné objekt nejenom klasifikovat, ale také detekovat. Vstupem je jeden obrázek s komplexní scénou. Výstupem jsou ohraničené objekty s predikovanou třídou.

Dále se bude práce zabývat sestavením datasetu, který poslouží pro trénování modelů neuronových sítí. Bude navržen a prakticky proveden postup tvorby datasetu pomocí metody reverzního vyhledávání obrázků.

Bude proveden návrh modulárního programu pro detekci a klasifikaci létajících objektů. Předpokladem je běh v reálném čase, vstupem je série chronologicky řazených snímků. Program bude obsahovat tři základní moduly: detekce, trasování a klasifikace. První modul označuje oblasti s předpokládaným objektem, modul trasování sleduje detekovaný objekt i na dalších snímcích a modul klasifikace bude rozpoznávat třídu tohoto objektu. Předpokladem je použití klasifikační neuronové sítě.

Druhým přístupem k řešení je použití modelu neuronové sítě, která detekuje a klasifikuje objekty na základě jednoho obrázku. Z teoretického úvodu bude vybrán jeden model z této kategorie.

Následně budou oba přístupy vyzkoušeny na testovacích datech. Výsledky budou okomentovány a vyhodnoceny.

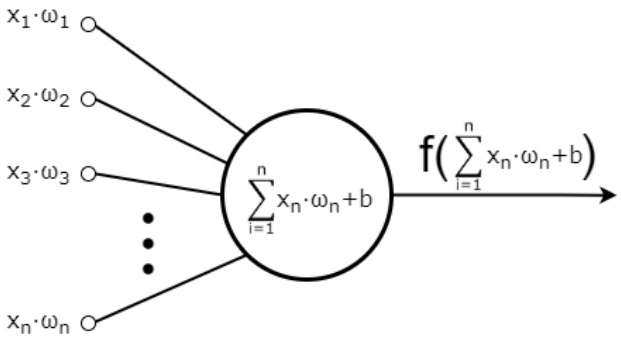
V poslední kapitole bude provedena rešerše a návrh hardwaru.

1. KONVOLUČNÍ NEURONOVÉ SÍTĚ

Ve strojovém učení je architektura neuronových sítí inspirována biologickým nervovým systémem. Neuronové sítě mohou být použity pro řešení obsáhlé řady komplexních úloh. Pro úlohy s vizuálním vstupem jsou primárně používány konvoluční neuronové sítě.

1.1 Model neuronu

Vstupem umělého neuronu může být libovolné množství signálů. Na vstup neuronu lze připojit vstupní data nebo výstup z jiného neuronu. Výstupem je vždy pouze jeden signál. Model neuronu je popsán na obrázku 1-1.



Obr. 1-1 Model umělého neuronu

Pro vyjádření důležitosti jednotlivých vstupů jsou použity váhy. Jakmile je spočtena suma za pomocí vstupů a vah, je nutné použít aktivační funkci. Pomocí aktivační funkce je simulovaná odezva biologického neuronu na vstupy.

1.1.1 Aktivační funkce neuronu

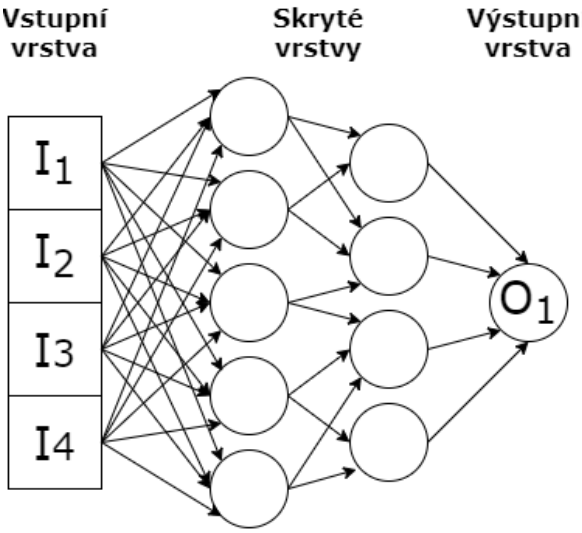
Pro výpočet lze použít širokou škálu funkcí. V tabulce 1-1 jsou vypsány běžně používané funkce. Populární je například použití *ReLU*. Tato funkce je výpočetně velmi nenáročná. Nevýhodou je však nerozlišování hodnoty u záporných čísel a výstup neuronu není vycentrován okolo nuly. Pro omezení nevýhod *ReLU* je možné použít *LReLU*, která přímou úměrou definuje výstupy pro záporné vstupy. Zajímavou funkcí je také *Maxou*t. Výstupem je maximální hodnota z jednotlivých vstupů neuronu.

Tabulka 1-1 Aktivační funkce

Název aktivační funkce	Matematický předpis
Sigmoid	$\boldsymbol{\sigma(x) = \frac{1}{1 + e^{-x}}}$
Hyperbolický tangens	$\boldsymbol{f(x) = tanh(x)}$
ReLU	$\boldsymbol{f(x) = max\,(0, x)}$
LReLU	$\boldsymbol{f(x) = max\,(0.01x, x)}$
Maxout	$\boldsymbol{f(x) = max\,(\omega_1 \cdot x + b_1, \omega_2 \cdot x + b_2, \ldots)}$

1.2 Umělá Neuronová síť

Vstupem umělé neuronové sítě je zpravidla vícerozměrný vektor. Tento vektor je vložen do vstupní vrstvy. Architektura většinou obsahuje i několik skrytých vrstev, které dodávají komplexnost výslednému modelu. Výstupní vrstva předává výsledek. Jednotlivé vrstvy mohou být různě propojeny. Často se používá plné propojení (*Fully connected*), při kterém všechny výstupy neuronů z předchozí vrstvy vstupují do všech neuronů vrstvy následující. Jednoduchý příklad umělé neuronové sítě je na obrázku 1-2.



Obr. 1-2 Příklad umělé neuronové sítě

1.3 Obecná Konvoluční neuronová síť

Jedná se o druh umělé neuronové sítě, která je uzpůsobena k práci s vizuálním vstupem. Rozdílem je použití konvolučních filtrů.

1.3.1 Vstup konvoluční neuronové sítě

Vizuální vstup je digitální obrázek. Obrázek má vždy určité rozměry (délka × výška), dále záleží na barevném zpracování. První varianta je černobílý obrázek. Každý pixel má jednu hodnotu. Nejčastěji se používá barevný model *RGB*. Každý pixel obsahuje tři

hodnoty. Z obrázku lze vyčíst větší množství informací, ale roste výpočetní náročnost modelu.

1.3.2 Konvoluční vrstva neuronové sítě

Úkolem této vrstvy je pomocí skalárního součinu s váhami prohledávat malé oblasti obrázku.

1.3.3 Skryté vrstvy neuronové sítě

Konvoluční neuronové sítě zpravidla obsahují alespoň jednotky až desítky skrytých vrstev. Význam skrytých vrstev si lze ukázat na představě nejjednodušší obecné neuronové sítě, která bude postupně modifikována. Začátkem je model, který má na vstupu vektor o *n* složkách a výstup tvoří pouze jedna hodnota. Není uvažována žádná skrytá vrstva, aktivační funkcí je $f(x) = x$. Tato aktivační funkce se v praxi nepoužívá, ale je vhodná pro tento příklad. Výstup modelu je potom lineárně závislý na vstupu a lze jej vyjádřit ve tvaru $y = \omega_1 \cdot x_1 + \ldots + \omega_n \cdot x_n$. Pro takto jednoduchou závislost se samozřejmě definovat neuronovou síť nevyplatí. Avšak přidáním skrytých vrstev je získána nelineární závislost, ale hlavně také komplexita celého modelu.

Skryté vrstvy jsou pro konvoluční neuronové sítě nepostradatelné, protože v první vrstvě zpravidla prohledáváme malé oblasti pomocí konvolučních filtrů. Skryté vrstvy následně využívají informace získané z jednotlivých částí celku.

Fully connected layer

Tato vrstva je plně propojená s vrstvou předchozí. Což znamená, že každý neuron z této vrstvy má na vstupu připojeny všechny neurony z vrstvy předchozí.

Pooling layer

Tato vrstva má zpravidla menší rozměry než vrstva předchozí, protože hodnota jednoho neuronu je spočtena z více vstupních neutronů předchozí oblasti, zpravidla ze čtvercové oblasti. Výsledná hodnotu příslušící tomuto neuronu je buď průměr nebo maximální hodnota z definované oblasti.

Dropout layer

Slouží proti přetrénování modelu. Náhodně jsou určeny nulové hodnoty ve vstupní vrstvě. Použití je pouze při trénování modelu.

Activation layer

V této vrstvě jsou použity neurony s aktivační funkcí. Příklady a popis aktivačních funkcích lze nalézt v podkapitole 1.1.1.

1.3.4 Výstup konvoluční neuronové sítě

Můžeme provést rozdělení *CNN* podle počtu výstupních hodnot. V prvním případě je výstupem pouze jedna hodnota a v druhém případě je výstup vektor o *n* prvcích. Dále je nutné určit implementaci výsledků. Například u klasifikační úlohy, kdy každý prvek výstupního vektoru vypovídá o pravděpodobnosti o příslušnosti obrázku k dané třídě, lze vybrat pouze nejvyšší hodnotu a tento obrázek přiřadit k této třídě. Druhou možností je

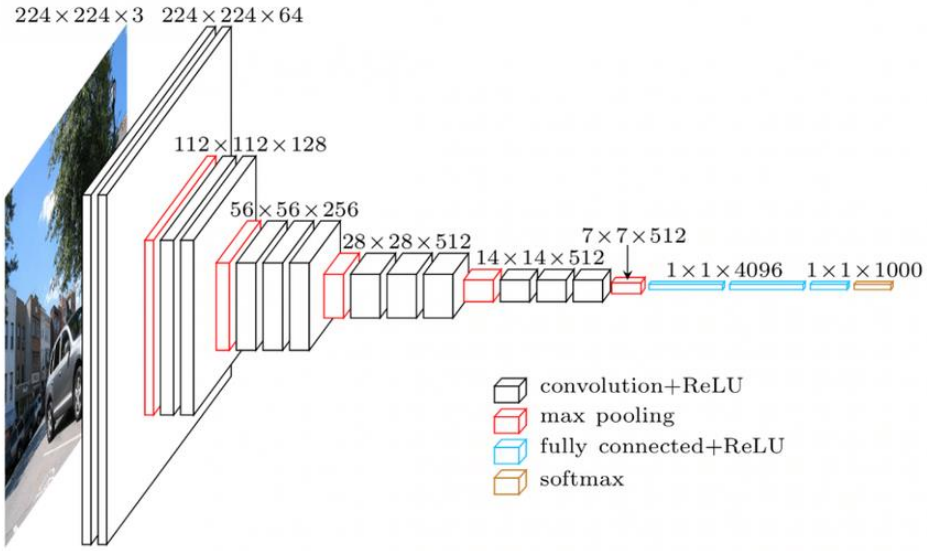
zobrazit všechny výsledky s odhadem pravděpodobnosti, nebo popřípadě zobrazit třídy s pravděpodobností nad určitou úrovní.

1.4 Modely konvolučních neuronových sítí pro klasifikaci

V současné době je k dispozici velká nabídka předtrénovaných modelů pro klasifikaci obrázků. Modely jsou trénovány na obsáhlých obecných datasetech. Počty obrázků pro jednu třídu se pohybují v řádu tisíců. Tyto modely lze přizpůsobit i pro vlastní trénovací datasety. Pro použití stačí změnit počet výstupních tříd. Výhodné je i v tomto případě použití předtrénovaných modelů, následně je model dotrénován na vlastních datech. Dále jsou uvedeny příklady modelů.

1.4.1 VGG

Přínosem je zkoumání vlivu počtu vrstev *CNN* na klasifikační výsledky. Výsledkem práce je stanovení optimálního počtu vrstev na šestnáct až devatenáct. Z toho vychází dvě používané varianty *VGG16* a *VGG19*. Specialitou modelu je použití velmi malých konvolučních filtrů (3 × 3). [1]



Obr. 1-3 Architektura modelu VGG16 [2]

1.4.2 Inception v1

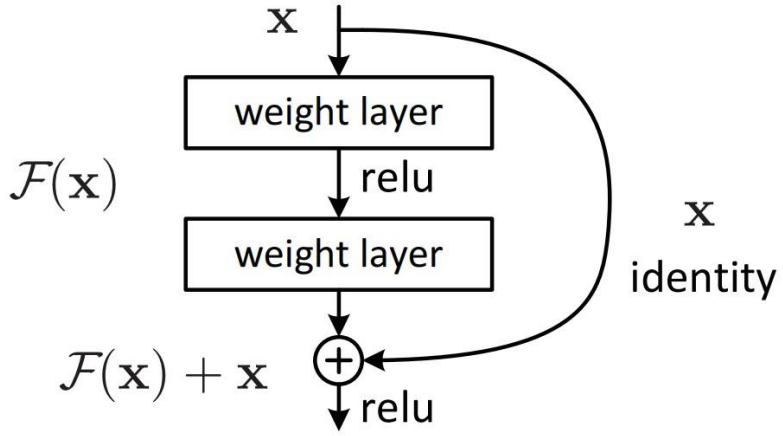
Model se vyznačuje optimalizovaným využitím výpočetního výkonu uvnitř neuronové sítě. Důvodem je pečlivě zvolená architektura, což umožnilo navýšit počet vrstev a zvětšit jejich šířku. [3]

1.4.3 ResNet

Model byl vytvořen z důvodu složitého trénování *CNN* s velkým počtem vrstev, řádově desítky až stovky vrstev. Z teorie konvolučních sítí vyplývá, že při navýšení počtu vrstev dochází k zachování, nebo zvýšení výkonosti modelu. Důvodem je možný přenos

nezměněných informací mezi jednotlivými vrstvami. Více vrstev však znamená větší komplexitu celého modelu a při trénování se může model dostat do lokálního optima.

Z důvodu popsaných v předchozím odstavci, obsahuje architektura modelu *ResNet* místa s přímým propojením mezi jednotlivými vrstvami. Z tohoto důvodu nemůže být informace z předchozích vrstev „ztracena“. [4]



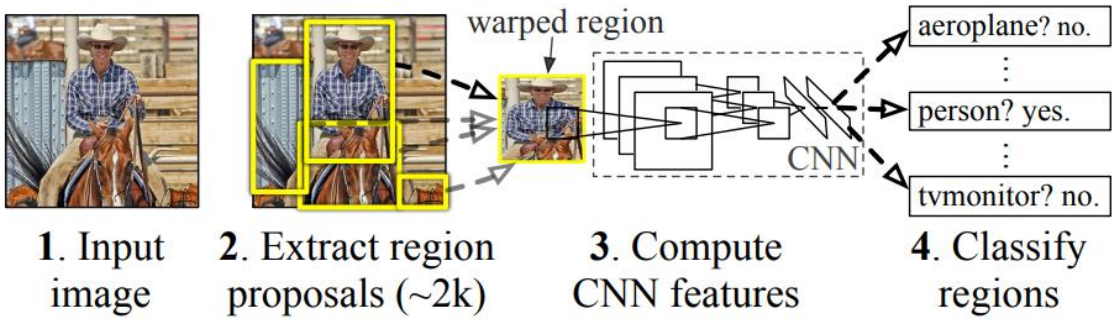
Obr. 1-4 Příklad přímého propojení mezi vrstvami sítě [4]

2. MODELY PRO DETEKCI A KLASIFIKACI OBJEKTŮ Z JEDNOHO SNÍMKU

Klasická *CNN* posuzuje obrázek jako celek a následný výstup může být klasifikován jako zařazení objektu k příslušné třídě nebo třídám. Pokud ale prvotním úkolem je detekce objektu, je nutné nejprve označit oblast na které se objekt nachází a následně jej klasifikovat. Z tohoto důvodu nabírá úloha na komplexitě a složitosti. V této kapitole jsou uvedeny a popsány nejvýznamnější modely.

2.1 R-CNN

V roce 2014 bylo dosaženo průlomu v identifikaci a klasifikaci objektů. *R-CNN* model lze rozdělit do tří částí. V první části jsou vytipovány oblasti, ve kterých je předpokládána přítomnost hledaných objektů. Druhá část používá *CNN* pro postupné vyhodnocení všech vytipovaných oblastí. Výstupem *CNN* je vektor. Ve třetí části je provedena klasifikace pomocí lineárního *SVM* modelu. Výsledkem může být přiřazení třídy objektu, nebo může být oblast vyřazena. Počet vytipovaných a následně klasifikovaných oblastí lze nastavit, ale nejčastější nastavení je dva tisíce. Tyto oblasti se můžou i překrývat, ale už v první části jsou odstraněny duplikované oblasti. Z modelu *R-CNN* vychází další modely *Fast R-CNN* a *Faster R-CNN*. [5]



Obr. 2-1 Princip funkce modelu R-CNN [5]

2.1.1 Vytipování oblastí

Teoretický model *R-CNN* neobsahuje striktní definici pro algoritmus vytipování oblastí pro klasifikaci. Může být použít jakýkoliv postup, který označí určitý počet oblastí. Pro různé aplikace modelu může být optimální algoritmus různý. Například v původní práci bylo pro demonstraci modelu *R-CNN* použito selektivního vyhledávání [6]. Vytipované oblasti jsou následně prověřeny pomocí *CNN*. Výstupem *CNN* je vektor s pevnou délkou. Vektor je následně použit jako vstup *SVM* klasifikátoru. [5]

Selektivní vyhledávání

Cílem vyhledávání je nalezení oblastí s možným výskytem objektů vhodných k identifikaci. Rozměry hledaných objektů v pixelech není známá, je proto vhodné použít

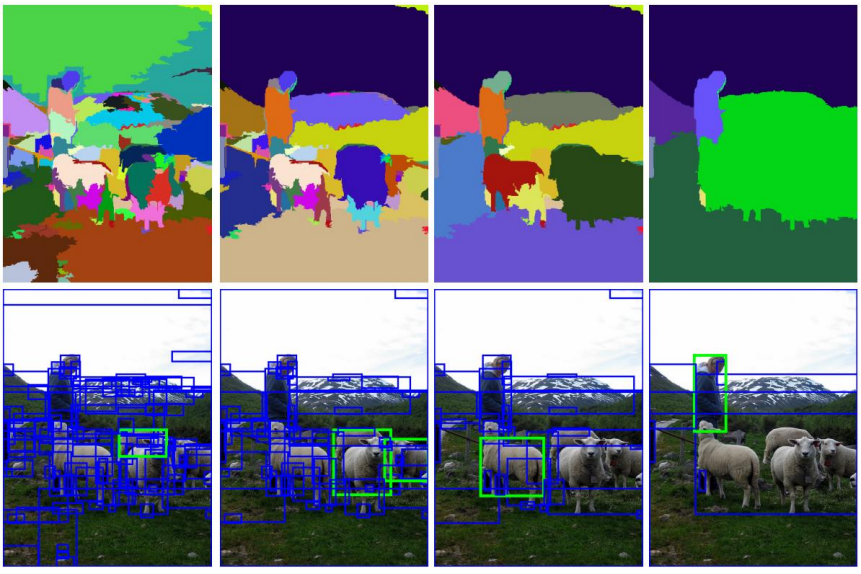
proměnnou velikost vytipovaných oblastí.

Prvním krokem je rozdělení celého obrázků na výchozí oblasti. Tyto oblasti obsahují objekty s malými rozměry v pixelech, jednoduché objekty, části objektů nebo pouze pozadí. Dále je nutné najít oblasti s komplexnějšími objekty, které se můžou například skládat z více jednodušších objektů, nebo byl objekt rozkouskován v prvním kroku. Jako příklad komplexnějšího objektu může být uvedena motorka, která se skládá z více dílů. Například je možné identifikovat kola a následně i motorku jako celek. Objekt může být chybně rozkouskován například vlivem nehomogenního osvětlení. [6]

Pro odhalení komplexnějších a „pixelově“ rozměrnějších objektů, je vhodným řešením nasazení hierarchického algoritmu, který postupně spojuje jednotlivé části obrázků do jednoho celku. Vyvstává otázka, jestli je vhodné zkoušet klasifikovat všechny oblasti vzniklé v průběhu vyhledávání, popřípadě kolik a jaké oblasti použít. [6]

V tomto odstavci je uvedena obecná procedura shlukování. Prvním krokem je vytvořit počáteční oblasti. Dále jsou jednotlivé oblasti spojovány dohromady pomocí hladového algoritmu. Iterace začíná spočtením podobnosti mezi sousedními oblastmi. Dvě nejpodobnější oblasti jsou spojeny dohromady a vzniká jedna nová oblast. Algoritmus je ukončen, jakmile jsou všechny původní oblasti sloučeny. Zpravidla předpokládaná důležitost oblasti roste s rostoucím počtem iterací. [6]

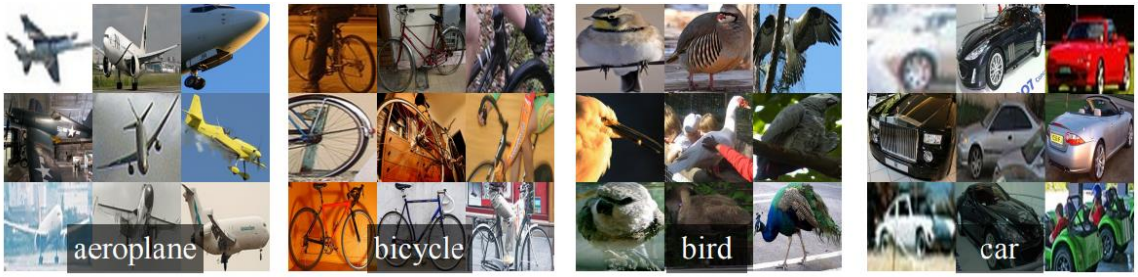
Pro splnění úlohy je použito více postupů určení výchozích oblastí a více druhů funkcí, kterými určujeme podobnost sousedních oblastí. Zároveň je nutné zaručit, aby rychlost vytipování oblastí byla dostatečně rychlá v kontextu celého modelu. Přípustná výpočetní náročnost také závisí na konkrétní aplikaci. [6]



Obr. 2-2 Ukázka postupu selektivního vyhledávání [6]

2.1.2 Analýza oblasti pomocí CNN

Pro vytvoření modelu *CNN* byl použít *open source framework Caffe* [7]. Implementace *CNN* byla provedena podle *ConvNet* popsané v práci [8]. Z orámované oblasti je vždy získán vektor s 4096 dimenzemi. Vstupem je obrázek o rozměrech 227 × 227 v barevném modelu *RGB*. Samotná *CNN* obsahuje 5 konvolučních vrstev a dvě plně propojené vrstvy. Vytipované oblasti většinou nejsou rozměru 227 × 227. Proto je nutné všechny vstupní oblasti na tento rozměr upravit. V práci [6] bylo požadovaných rozměrů docíleno nejjednodušší cestou. Vytipované oblasti byly přepočítány na požadovanou oblast 227 × 227. Před samotným přepočtem bylo původní orámování rozšířeno o konstantní počet pixelů *p* do všech čtyř stran. Konkrétně bylo použito *p = 16*. [5]



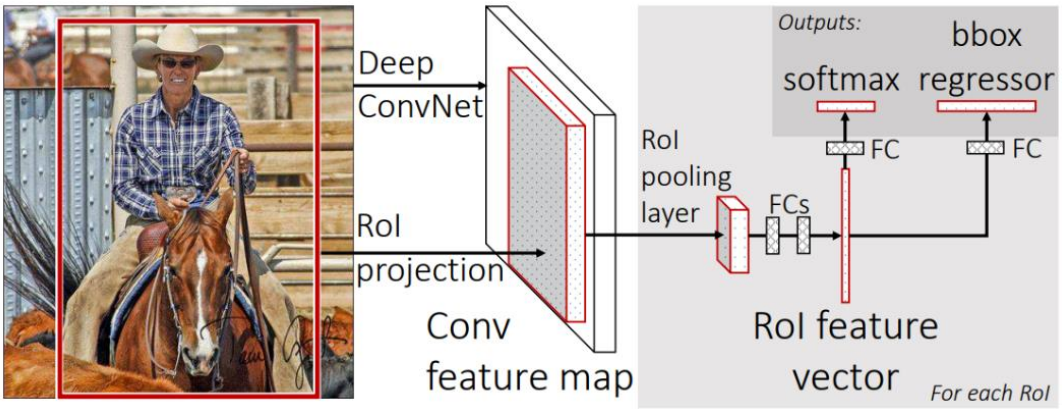
Obr. 2-3 Snímky s pevnými rozměry z trénovacího datasetu R-CNN [5]

2.1.3 Klasifikace pomocí SVM

Vstupem klasifikátoru je výstupní vektor z *CNN*. Výstupem je třída. [5]

2.2 Fast R-CNN

Model je vylepšením původní *R-CNN* architektury. Je zkrácena doba pro trénování i testování modelu. Rovněž byla zvýšena úspěšnost klasifikace. Z toho vyplývá, že nová verze je lepší ve všech směrech. *R-CNN* je pomalejší, protože *ConvNet* provádí výpočty pro každou vytipovanou oblast zvlášť. Elegantnější je sdílet vhodné výpočty a zkrátit tak celkový čas počítání. Dále je model *R-CNN* trénován ve třech krocích. Prvním krokem je trénování části, která vytipovává oblasti. Druhý krok je *CNN*. Posledním krokem je *SVM* pro klasifikaci objektů do tříd. Model Fast *R-CNN* tento nedostatek odstraňuje a trénování probíhá v jednom kroku. [9]



Obr. 2-4 Princip funkce modelu Fast R-CNN [9]

2.2.1 Konvoluční funkční mapa

Tato část obsahuje několik konvolučních a *max pooling* vrstev. Vstupem je celý obrázek. Výstupem je konvoluční funkční mapa, která je dále zpracovávána. Výhodou je použití jednoho výpočtu pro všechny vytipované oblasti. [9]

2.2.2 Vrstva pro vytipované oblasti

V této vrstvě jsou použity max pooling vrstvy. Informace z vytipovaných oblastí byly nejprve transformovány do konvoluční funkční mapy a následně jsou v této části opět transformovány do nové podoby. Nová funkční mapa má pevné rozměry $H \times W$. Například v práci [9] je uváděn jako příklad rozměr 7×7. Tento rozměr může být měněn, ale platí stejný rozměr pro všechny vytipované oblasti. Každá vytipovaná oblast je vyznačena pomocí čtyř čísel (r,c,h,w). Levý horní roh vytipované oblasti udává první dvojice (r,c). Druhá dvojice označuje výšku a šířku oblasti (h,w).

2.2.3 Funkční vektor vytipované oblasti

Vstupem je funkční mapa vytipované oblasti s pevným rozměrem určeným v předchozí části ($H \times W$). Výstupem je vektor, který slouží ke klasifikaci objektu. Zároveň je tento vektor použit pro úpravu ohraničení oblasti. Pro tuto úlohu je vhodné použít předtrénovanou neuronovou síť. V práci [9] je uvedena *VGG16*. Použita je část s plně propojenými vrstvami.

2.2.4 Výstupní třída

Výstupem je klasifikace objektu do třídy, nebo označení oblasti jako pozadí. Pro každou třídu je odhadnuta pravděpodobnost výskytu na dané oblast. Obsahuje plně propojenou vrstvu a *softmax vrstvu*. Klasifikace je v práci [9] prováděna pomocí klasifikátoru *ImageNet*.

2.2.5 Změna ohraničení oblasti

Používá stejný vstup jako výstupní část pro klasifikaci třídy, což je vhodné z hlediska

výpočetní náročnosti. Pro všechny natrénované třídy jsou vypočteny nové hranice oblasti zvlášť. Jedná se o pozici levého horního rohu, šířku a výšku oblasti, jak již bylo uvedeno dříve v této kapitole. [9]

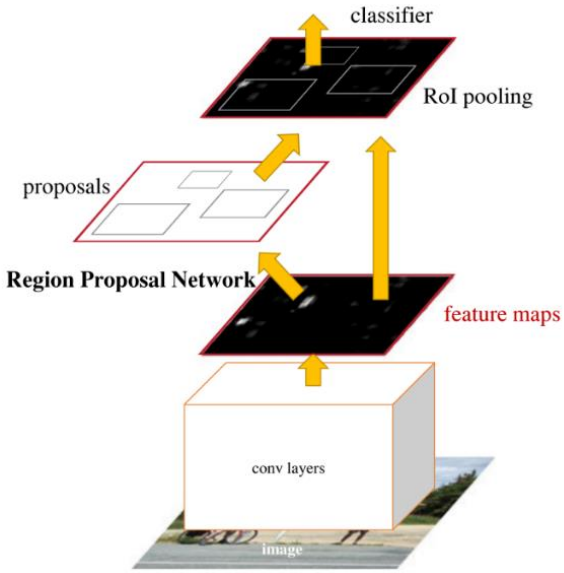
2.2.6 Trénování

Pro zrychlení trénovaní modelu oproti minulé verzi *R-CNN* jsou při trénování sdíleny výpočty. Konkrétně je použit *Stochastic gradient descent* (*SGD*) a *mini batches*. Pro trénování je nutné zvolit *N*, které značí počet obrázků a *R* označující počet vytipovaných oblastí (pro všechny obrázky, pro jeden obrázek je počet vytipovaných oblastí *R/N*). Výpočty pro vytipované oblasti na jednom obrázku jsou sdíleny. V práci [9] je uveden příklad použití $N = 2$ a $R = 128$ a porovnání rychlosti trénování s použitím 128 oblastí z různých obrázků. Trénování s použitím různých obrázků bylo zhruba 64 - krát pomalejší. Nabízí se otázka, jestli nenastane problém s konvergencí modelu při použití výše popsaného postupu. Důvodem je menší pestrost výběrů oblastí. Tyto obavy se však příliš nepotvrdily, kdy při nastavení $N = 2$ a $R = 128$ nebyl problém model trénovat. [9]

Zrychlení bylo docíleno také pomocí jednokrokového trénování, kdy nejsou postupně trénovány jednotlivé části modelu jako v případě R-CNN. V případě tohoto modelu je počítaná ztrátová funkce pro model jako celek. [9]

2.3 Faster R-CNN

Předchozí modely *R-CNN* a *Fast R-CNN* potřebují pro svoji činnost externí algoritmus, který vytipovává oblasti, ve kterých očekává objekty pro klasifikaci. Tato část se stala úzkým hrdlem celého modelu z pohledu výpočetní náročnosti. Nový model přichází s *Region Proposal Network* (*RPN*), což je konvoluční neuronová síť pro detekci a označení objektů ve vstupním obrázku. [10]



Obr. 2-5 Princip funkce modelu Faster R-CNN [10]

2.3.1 RPN

Vstupem je obrázek o libovolných rozměrech. Výstupem jsou ohraničené oblasti s bodovým ohodnocením. *RPN* a *CNN* pro klasifikaci sdílí konvoluční vrstvy, což znamená, že místo dvou výpočtů je proveden pouze jeden. [10]

Vytipované oblasti jsou vytvořeny při procházení konvoluční funkční mapy pomocí malé neuronové sítě. Síť je postupně posunovaná po celé mapě. Vstupem sítě je okno s rozměrem $n \times n$. Například v práci [10] je použito $n = 3$. Při prohledávání je střed okna nazýván *anchor*. Maximální počet vytipovaných oblastí je označován k. Pokud například jsou použity tři různá měřítka a tři různé poměry stran, $k = 9$. Určení oblasti je provedeno pomocí čtyř čísel, takže ohraničení má až $4 \times k$ údajů. Na oblasti je nebo není objekt. Klasifikace odpovídá velikosti $2 \times k$. Celkový počet *anchor* je vypočítán také na základě rozměru konvoluční funkční mapy. Celkový počet je WHk. Velikost výstupní vrstvy lze spočítat jednoduše $(4 + 2) \times k$. Závorka odkazuje na označení vytipované oblasti a klasifikaci objektu.

2.3.2 Trénování RPN

Při trénování jsou všechny oblasti (*anchor*) rozděleny do dvou skupin. Jedná se o binární rozdělení na objekty a pozadí. Do skupiny objekty jsou zařazeny oblasti s nejlepším překrytím se vzorovým orámováním objektu a také oblasti u kterých je *Jaccard index* vyšší než 0.7. [10]

Výpočet pro *Jaccard index*:

$$J(A, B) = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}, \tag{2.1}$$

kde A je vzorové orámování objektu a B je anchor oblast.

2.3.3 Klasifikace objektů

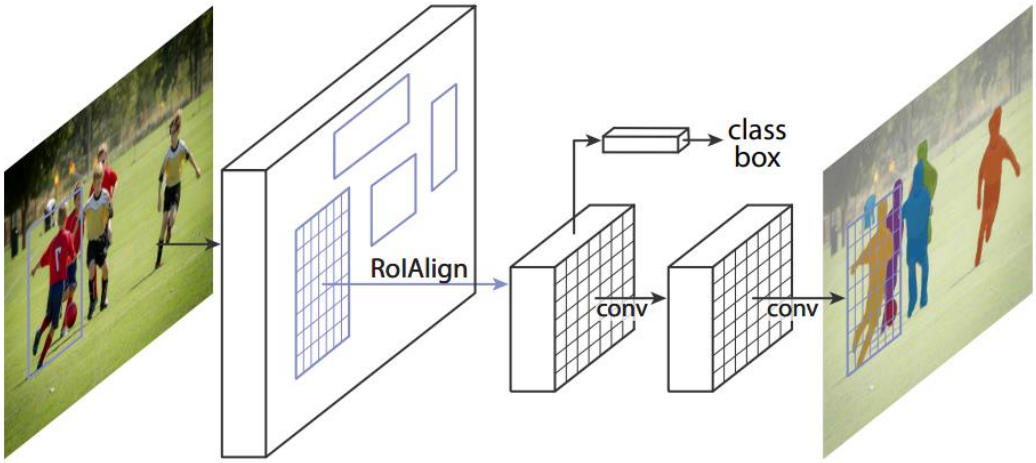
Výstupní vektor je stanoven pro všechny vytipované oblasti pomocí *RPN*. Jsou použity výpočty z konvolučních vrstev *RPN* modulu. Pro samotnou klasifikaci jsou používány *CNN* podle preference. V práci [10] je například použita *VGG16* nebo *ZF* síť, ale i mnoho dalších.

2.4 Mask R-CNN

Nový model z řady *R-CNN* opět vylepšuje předcházející modely. *Mask R-CNN* se vydal cestou větší flexibility a robustnosti než předchozí modely. Předchozí modely jsou rozděleny na část segmentace a klasifikace. Nový model je založen na paralelní předpovědi masky a označení třídy, což je jednodušší a flexibilnější. [11]

Faster R-CNN má dva výstupy, třídu objektu a orámování oblasti objektu. Mask R-CNN přidává třetí část, která obsahuje masku objektu. Pro získání masky objektu je nutné daleko jemnější prohledávání oblasti než u předchozích modelů. Je nutné se dostat až

k samotnému pixelovému uspořádání obrázku. Princip modelu a ukázka masky objektu jsou zobrazeny obrázku 2-6. [11]



Obr. 2-6 Princip funkce modelu Mask R-CNN [11]

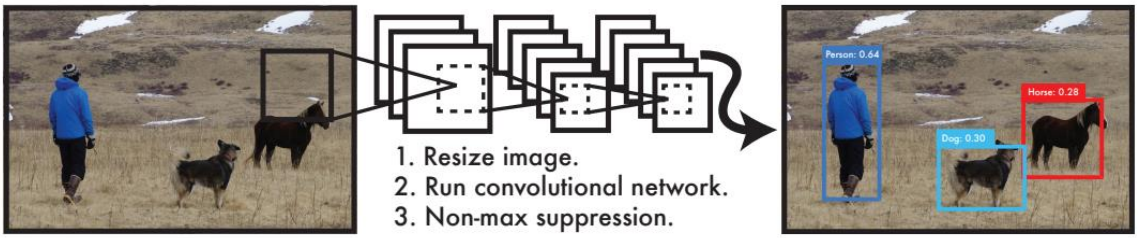
2.4.1 Maska objektu

Pro každou oblast z první části je vytvořena $m \times m$ maska, kde rozměr m je totožný rozměr s oblastí. Každý pixel obsahuje hodnotu „1“ nebo „0“. Takto je oblast rozdělena na objekt a pozadí. [11]

2.5 YOLO

Model je vhodný pro aplikace v reálném čase. Ačkoliv model nedosahuje přesnosti modelů popsaných výše, pro spoustu aplikaci je klíčová rychlost.

Model je tvořen pouze jednou neuronovou sítí. Orámování a klasifikace objektu je provedena v jednom kroku. Princip zpracování obrázků je jednoduchý, nejprve je vstupní obrázek převeden na rozměry 448 × 448 pixelů, dále je spuštěna konvoluční neuronová síť a v posledním kroku jsou zobrazeny detekované objekty s vyšší výstupní hodnotou, než je hodnota požadovaná. Výstupní hodnota označuje důvěru modelu ve správnou detekci objektu. [12]

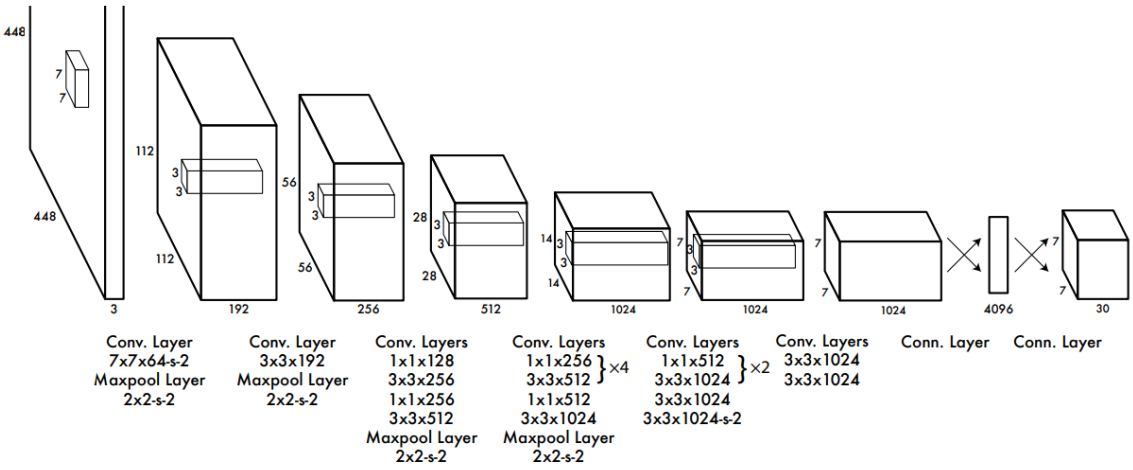


Obr. 2-7 Princip funkce modelu YOLO [12]

Architekturu konvoluční neuronové sítě lze rozdělit na dvě částí. První část tvoří konvoluční vrstvy. Tato část získává funkční mapy z obrázků. Druhá část obsahuje plně

propojené vrstvy. Zde dochází k odhadu pravděpodobnosti výskytu objektu a určení jeho orámování. [12]

Konkrétně je architektura inspirována modelem *GoogLeNet* [13]. Model *YOLO* obsahuje dvacet čtyři konvolučních vrstev a dvě plně propojené vrstvy. Byla vytvořena i varianta s pouze devíti konvolučními vrstvami, která se nazývá Fast YOLO. Architektura je blíže ukázaná na obrázku 2-8. [12]



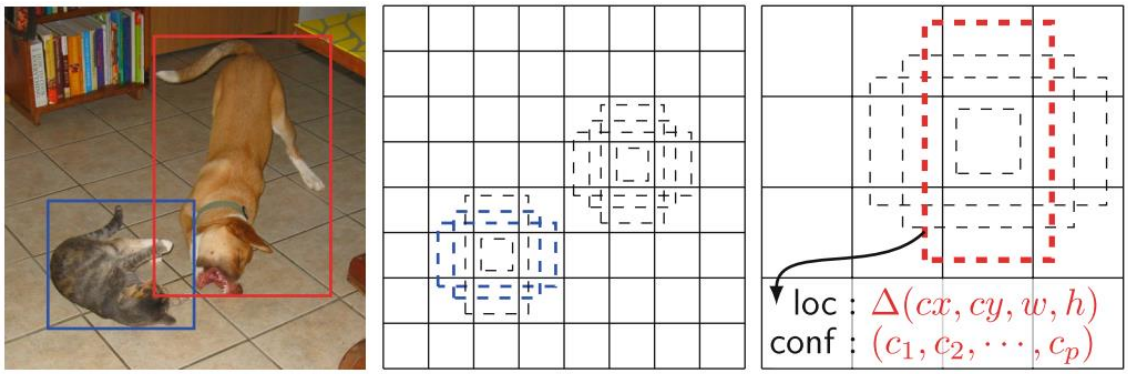
Obr. 2-8 Struktura modelu YOLO [12]

Mezi slabé stránky modelu patří špatné rozlišení objektů nacházejících se blízko u sebe, jako příklad můžeme uvést hejno letících ptáků. Dále má model problém s generalizací objektů, které se vyskytují v neobvyklém novém poměru stran nebo jiné konfiguraci. Vlivem možné změny rozlišení vstupních obrázku může být orámování oblasti nepřesné. Při trénování je chyba orámování počítaná stejně pro malé a velké objekty, přitom malá chyba u velkého objektu je zanedbatelná, ale u malého objektu i malá chyba může způsobit problémy. [12]

2.6 SSD

SSD model je reakcí na poptávku po výpočetně méně náročném modelu pro detekci a klasifikaci. Tento model je vhodný pro použití v *embedded* systémech a pro aplikace běžící v reálném čase. [14]

Model *SSD* detekuje objekty na základě jednoho snímku. Výsledkem je podobná přesnost jako u modelu *Faster R-CNN*, ale výpočet je rychlejší, což je hlavní výhoda. Algoritmus je založen na prohledávání oblasti pomocí rámečků s fixními rozměry. Každá oblast následně dostává skóre, které je úměrné pravděpodobnosti přítomnosti objektu. Skóre je stanoveno pro každou třídu zvlášť. Následně je z těchto dat rozhodnuto o konečné klasifikaci. [14]

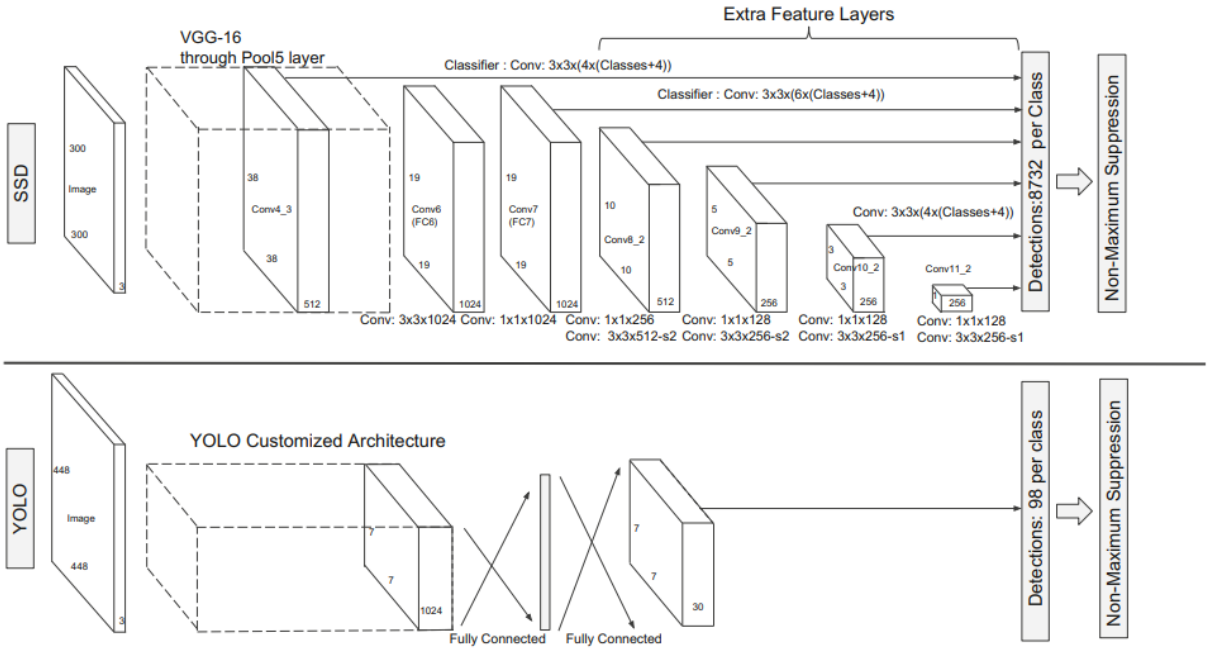


(a) Orámované objekty (b) 8×8 funkční mapy (c) 4×4 funkční mapy

Obr. 2-9 Princip funkce modelu SSD [14]

Model obsahuje konvoluční vrstvy, které jsou napojeny tak, že se rozměry těchto vrstev postupně snižují. Proto je možné model použít na různé rozlišení obrázků. Každá konvoluční vrstva může generovat pevný počet detekcí. [14]

SSD lze přirovnat k *YOLO* modelu. Výhodou obou modelů je jednoduchý princip a struktura modelu. Další výhodou je řádově rychlejší zpracování obrázků než u modelů z řady *R-CNN*. Nevýhodou je o něco menší přesnost. Na obrázků 2-10 lze vidět porovnání *SSD* a *YOLO* modelu. Konkrétně se jedná o model *SSD300*. Modely se liší potřebným rozlišením obrázků a celkově je uspořádání skrytých vrstev odlišné, ale základní princip je u obou modelů stejný. [14]



Obr. 2-10 Porovnání architektury modelu SSD a YOLO [14]

2.7 Shrnutí

Modely lze rozdělit do dvou kategorií. Do první kategorie lze zařadit modely typu *R-CNN*. Tyto modely obsahují vcelku složitou architekturu, rychlost zpracovávání vstupních snímků je poměrně pomalá. Počet zpracovaných snímků závisí také na výpočetním výkonu, ale lze očekávat maximální hodnoty kolem jednotek za sekundu.

Druhou kategorii tvoří modely typu *SSD* a *YOLO*. Rychlost zpracovávání obrazu je na prvním místě. Architektura modelu je podstatně jednodušší.

Pokud se zaměříme na modely pro aplikaci v reálném čase, například model *YOLO* je velmi populární a vznikla řada verzí. Poslední verze jsou *YOLOv4* a *YOLOv5*. Jsou zde uvedeny dvě verze, protože pojmenování modelu *YOLOv5* je kontroverzní. *YOLOv5* byl vytvořen jiným týmem a neobsahuje příliš výrazný posun dopředu co se týče architektury. Na druhou stranu je model implementován v knihovně *PyTorch* a trénovaní vlastního modelu je uživatelsky přívětivé.

3. TVORBA VLASTNÍHO DATASETU

Nezbytnou součástí trénování modelu pro detekci a klasifikaci objektů je dataset s popisky. Model před trénováním si lze představit jako prázdnou nádobu. Znalost modelu je získaná až při učení na dodaných datech. Modely probrané v předchozích kapitách lze použít pro širokou paletu úloh. Například model může mít stejnou vnitřní architekturu pro klasifikaci dopravních prostředků na silnici nebo klasifikaci květů lučních květin. U obou uvedených příkladů lze dosáhnout vysoké přesnosti.

První možností je použít naučený model, který dokáže rozeznat desítky tříd a při tréninku byl použit kvalitní vstupní dataset. Tento přístup lze použít, pokud je potřeba detekovat a klasifikovat pouze běžné třídy, jako například člověk, auto, autobus atd. Těžší bude nalézt naučený model pro konkrétní úlohy, jako přiklad lze uvést výše zmíněnou klasifikaci lučních květin. Pro tyto úlohy je nutné provést trénování modelu na vlastním datasetu. Tato část nesmí být podceněna, protože bez dobrého datasetu nelze model správně natrénovat.

3.1 Výběr obrázků

Dataset musí obsahovat snímky všech tříd, které chceme model naučit a zároveň je doporučeno udržovat přibližně stejné početní zastoupení všech tříd. Další snahou je pokrýt co nejširší počet možných variant dané třídy. Například pokud budou mít všechny auta v datasetu pouze černou barvu, model nemusí klasifikovat správně auta jiné barvy. Zároveň je vhodné udržet variabilitu kompozice a pozadí snímků.

3.2 Popisky obrázků pro detekci a klasifikaci

Pro trénování je nutné označit a přiřadit třídu objektům, které má model rozpoznat. Popisky jsou zpravidla uloženy v samostatném dokumentu pro každý obrázek datasetu zvlášť. Pokud dataset obsahuje videa, lze vytvořit pouze jeden dokument pro všechny snímky daného videa.

```
"categories": [
    {
        "id": 0,
        "name": "Workers",
        "supercategory": "none"
    },
    {
        "id": 1,
        "name": "head",
        "supercategory": "Workers"
    },
```

(a) COCO JSON

```
<annotation>
        <folder></folder>
        <filename>000001.jpg</filename>
        <path>000001.jpg</path>
        <source>
                <database>roboflow.ai</database>
        </source>
        <size>
                <width>500</width>
                <height>375</height>
                <depth>3</depth>
        </size>
```

(b) Pascal VOC XML

```
1 0.617 0.3594420600858369 0.114 0.17381974248927037
1 0.094 0.38626609442060084 0.156 0.23605150214592274
1 0.295 0.3959227467811159 0.13 0.19527896995708155
1 0.785 0.398068669527897 0.07 0.14377682403433475
1 0.886 0.40879828326180256 0.124 0.18240343347639484
1 0.723 0.398068669527897 0.102 0.1609442060085837
1 0.541 0.35085836909871243 0.094 0.16952789699570817
1 0.428 0.4334763948497854 0.068 0.1072961373390558
1 0.375 0.40236051502145925 0.054 0.1351931330472103
```

(c) YOLOv5 TXT

Obr. 3-1 Ukázka různých formátů popisků [15]

Minimální obsah popisku je orámování objektu a přiřazení třídy. Na jednom obrázku může být vyznačeno libovolné množství objektů. Popisky mohou obsahovat i další dodatečné informace, záleží na jejich formátu.

3.3 Rozdělení datasetu

Nakonec je nutné připravit dataset s popisky k samotnému procesu učení. Většinou učení probíhá ve více iteracích, což jsou opakující se cykly, při kterých je snahou zvýšit přesnost modelu. Zpravidla se iterace skládá ze dvou částí. V první části model generalizuje poznatky z trénovacích dat a následně dochází k ověření změn modelu na validačních datech.

Z důvodů popsaných v předchozím odstavci, je nutné před samotným trénováním data rozdělit. Zpravidla se používá rozdělení na tři části, trénovací data, validační data a testovací data. V průběhu trénovaní se používají pouze trénovací a validační data. Snímky určené pro testování se nesmí používat při průběhu učení. Testovací data slouží pouze k určení přesnosti modelu.

3.4 Možné postupy při tvorbě datasetu

K úloze získání datasetu lze přistoupit více způsoby. Nejjednodušší metodou je vyhledávání již vytvořených datasetů. Popřípadě je možné spojit více různých datasetů dohromady. Dalším postupem je skládání vlastního datasetu. Časově náročný je proces vyhledávání i následná tvorba popisků, ale výsledný dataset odpovídá specifickým požadavkům úlohy.

V této práci byla tvorba datasetu realizována pomocí metody reverzního vyhledávání. Jako vstup pro vyhledávání je použit obrázek. Této části práce je věnovaná zvýšená pozornost, protože kvalitní dataset je nezbytný základ a předpoklad pro vznik robustního modelu. Postup tvorby datasetu je aplikovatelný pro různé úlohy detekce a klasifikace.

Pro tuto práci je potřeba vytvořit dataset pro detekci a klasifikaci, což je složitější než poskládat dataset pouze pro klasifikaci. Jsou kladeny vyšší nároky na pestrost datasetu a popis obrázků je náročnější. Pro klasifikační dataset stačí rozdělení snímků do tříd. Aby bylo možné natrénovat také detekci, je nutné objekty na obrázcích ohraničit a přiřadit třídu.

3.5 Testovací dataset

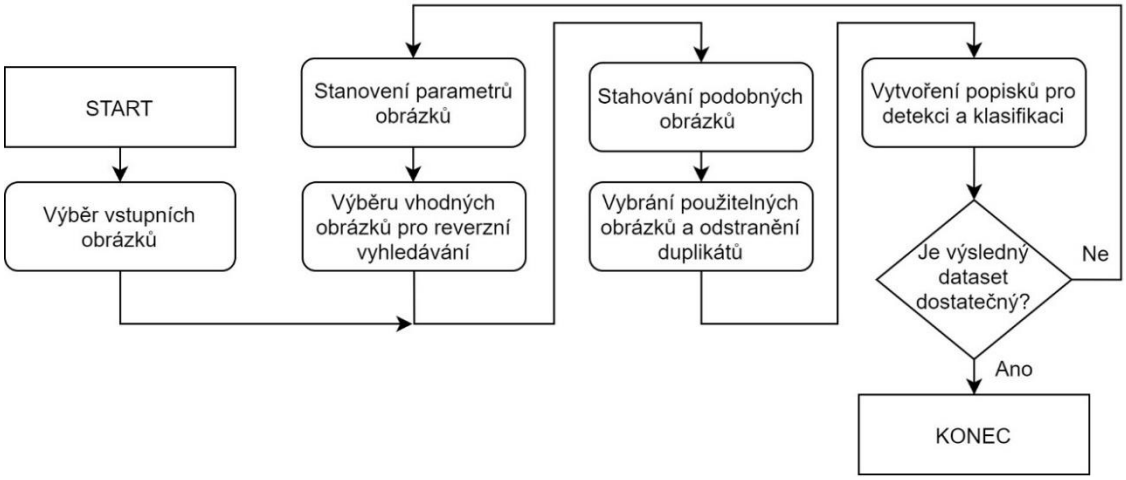
Cílem práce je vytvoření vlastního datasetu, ale je vhodné použít jiný dataset na testování. Důvodem je ověření robustnosti natrénovaného modelu. Jedná se i ověření trénovacích dat. Pokud je trénovací dataset dostatečně obsáhlý a pestrý, natrénovaný model bude fungovat i na nepatrně odlišných datech. Navíc výsledky budou odpovídat využití modelu v praxi. Konkrétně byl použit dataset z práce [16].

V daném testovacím datasetu jsou videa se čtyřmi různými třídami: letadla, ptáci, drony a helikoptéry. Celkově je obsaženo téměř tři sta videí o délce několika sekund. Rozlišení všech videí je 640 × 512 pixelů. Autorem jsou vytvořeny popisky v programovacím jazyce *MATLAB*. Formát popisku je *groundTruth* a je vytvořen samostatně pro každé video. Pro jednoduchost bude tento dataset označen číslem 285 (přesný počet videí). Na obrázku 3-2 je zobrazena ukázka snímků.

Obr. 3-2 Ukázka snímků z datasetu 285 [16]

3.6 Obecný postup s použitím reverzního vyhledávání

Postup vytvoření datasetu v této práci je popsán na obrázku 3-3. Hlavní myšlenkou je používání reverzního vyhledávání ve smyčce, dokud není dosaženo požadovaného obsahu datasetu. Jednotlivé části algoritmu jsou dále detailněji popsány.



Obr. 3-3 Popis tvorby datasetu pomocí reverzního vyhledávání

3.6.1 Výběr vstupních obrázků

Požadavkem je pestrost vstupních obrázků, protože nově nalezené snímky budou podobné těm vybraným. Před samotným vybíráním obrázků je vhodné stanovit, do jakých kategorií je možné obrázky členit. Například lze obrázky dělit podle počtu objektů v jednom snímku, podle poměrné oblasti zaujímané objektem a samozřejmě taky podle třídy objektu. Pro různé druhy datasetu je vhodné použití odlišných kategorií, které odpovídají požadavkům úlohy.

Výběr vstupních obrázků nemusí být příliš obsáhlý, ale je nutné, aby obsahoval co nejvíce kombinací kategorií, které jsou stanoveny před začátkem výběru. Tímto krokem je zaručena pestrost vstupních obrázků, což je předpoklad pro pestrost výstupního datasetu.

3.6.2 Stahování podobných obrázků

Pro získání nových obrázků pomocí metody reverzního vyhledávání, jsou používány webové prohlížeče. Nejdříve je na server nahrán obrázek, který slouží jako vstup. Výsledkem hledání jsou podobné obrázky. V této fázi je buď staženo *n* prvních zobrazených výsledků, nebo je možné spojit stahování se selekcí vybraných obrázků, v tomto případě jsou staženy pouze použitelné obrázky.

3.6.3 Vybírání použitelných obrázků a odstranění duplikátů

Jakmile je ukončen proces stahování, je nutné vyřazení obrázků, které jsou již obsaženy v datasetu. Dalším krokem je manuální vizuální kontrola a odstranění obrázků, které nejsou vhodné pro zařazení do datasetu.

3.6.4 Vytvoření popisků pro detekci a klasifikaci

Dalším krokem je vytvoření popisků obrázků. Popisky jsou ve formátu, který umožňuje trénování modelu. Popřípadě je možné tyto popisky jednoduše převést do jiného formátu.

Zároveň popisky slouží pro stanovení parametrů obrázků.

3.6.5 Stanovení parametrů obrázků

V tomto bodě jsou získány parametry obrázků, které slouží primárně jako kritérium pro výběr vhodných obrázků pro nové reverzní vyhledávání. Cílem je získat pestrý výběr, což je dosaženo vybíráním snímků s různými kombinacemi parametrů. Stanovení parametrů může být provedeno ručně nebo automaticky. Zpravidla je vhodné použít jednoduché automatické určení parametrů. Prvním důvodem je vysoká časová náročnost ručního stanovování parametrů. Dále už není provedení optimálního výběru obrázků tak kritické jako při vstupním výběru obrázků. Pro stanovení parametrů je vhodné použít i popisek obrázků. Z popisku obrázku lze v každém případě zjistit třídu objektu, počet objektů a vypočíst poměrnou plochu objektu. Dále je vhodné doplnit další parametry, které jsou relevantní pro daný dataset.

3.6.6 Výběr vhodných obrázků pro reverzní vyhledávání

Provedení výběru obrázků může být dosaženo automaticky pomocí požadavku na vytvoření datasetu a parametrů obrázků. Možností je i ruční výběr, ale opět nelze doporučit z důvodu časové náročnosti. Stanovené parametry obrázků lze použít i pro analýzu dosavadního složení datasetu. Na základě této analýzy a požadavků na vytvořený dataset, lze optimalizovat výběr vhodných obrázků pro reverzní vyhledávání. Například i přes pestrý výběr obrázků pro reverzní vyhledávání, může být dosavadní dataset nevyrovnaný. Jako příklad lze uvést menší poměrné zastoupení obrázků, které obsahují scénu s více objekty. Pomocí analýzy datasetu lze tento nedostatek odhalit. Při následujícím reverzním vyhledávání bude poměr snímků s více objekty navýšen.

3.7 Konkrétní postup pro danou úlohu

V této podkapitole je popsán proces tvorby datasetu pro úlohu detekce a klasifikace létajících objektů. Aby bylo splněno zadání úlohy, dataset musí obsahovat třídy: letadlo, pták, dron, helikoptéra a hmyz. Cílem bylo provést návrh datasetu s důrazem na využití vlastních automatických a poloautomatických skriptů v jazyce *pythonu*. Výhodou je úspora času oproti vytváření datasetu manuálně. Navíc lze kód poměrně jednoduše modifikovat pro vytvoření libovolného datasetu.

3.7.1 Výběr vstupních obrázků

Pro vytvoření vstupního výběru byly využity vybrané snímky z datasetu 285. Aby byl výběr dostatečně pestrý, byl doplněn ručně vyhledanými obrázky.

3.7.2 Stahování podobných obrázků

Byl použit webový prohlížeč *Yandex*. Alternativně lze použít také prohlížeče *Google* a *Bing*. Možností je i použití více prohlížečů kombinovaně. Například při požadavku na

větší počet dat. Při opakovaném reverzním vyhledávání totiž dochází k opakovanému stahování stejných obrázků. Změna prohlížeče může tento problém vyřešit. Prohlížeč *Yandex* byl vybrán na základě praktického zkoušení. Výsledky reverzního vyhledávání obsahovaly daleko vyšší procento použitelných obrázků než u alternativ.

Snahou bylo vždy stáhnout patnáct prvních obrázků z výsledků vyhledávání. Pokud nebylo možné obrázek stáhnout ve formátu *.jpg*, byl obrázek přeskočen. Většinou bylo staženo deset až čtrnáct snímků na každý vstupní obrázek. Stahování bylo provedeno ručně.

3.7.3 Vybírání použitelných obrázků a odstranění duplikátů

Prvním krokem byla vždy kontrola na přítomnost duplicitních obrázků. Nutné je srovnávat opravdu všechny obrázky, ne pouze snímky v dané iteraci. Lze tak předejít zbytečné práci s redundantními daty, které by byly stejně smazány při konečné kontrole. Kontrola byla provedena pomocí vlastního skriptu *remove_duplicate_files.py*. Skript kontroluje přítomnost identicky stejných obrázků a do nové složky zkopíruje pouze unikátní obrázky (u duplicitních pouze první). Skript ošetřuje i možnost kontroly duplicit až po vytvoření popisků a parametrů. V tomto případě jsou popisky a parametry překopírovány do nových složek současně s obrázkem.

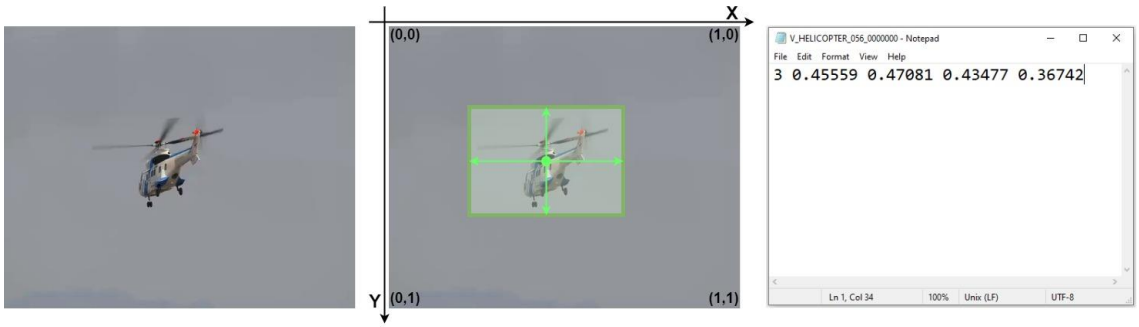
Samotné posouzení použitelnosti obrázku bylo provedeno manuálně. Byly odstraněny všechny obrázky, které neobsahovaly objekty požadované třídy nebo nebylo možné třídu objektu určit.

3.7.4 Vytvoření popisků pro detekci a klasifikaci

Popisky obrázků byly tvořeny ve formátu *YOLO*. Popis obsahuje libovolný počet objektů a každý objekt je označen třídou a ohraničením oblasti.

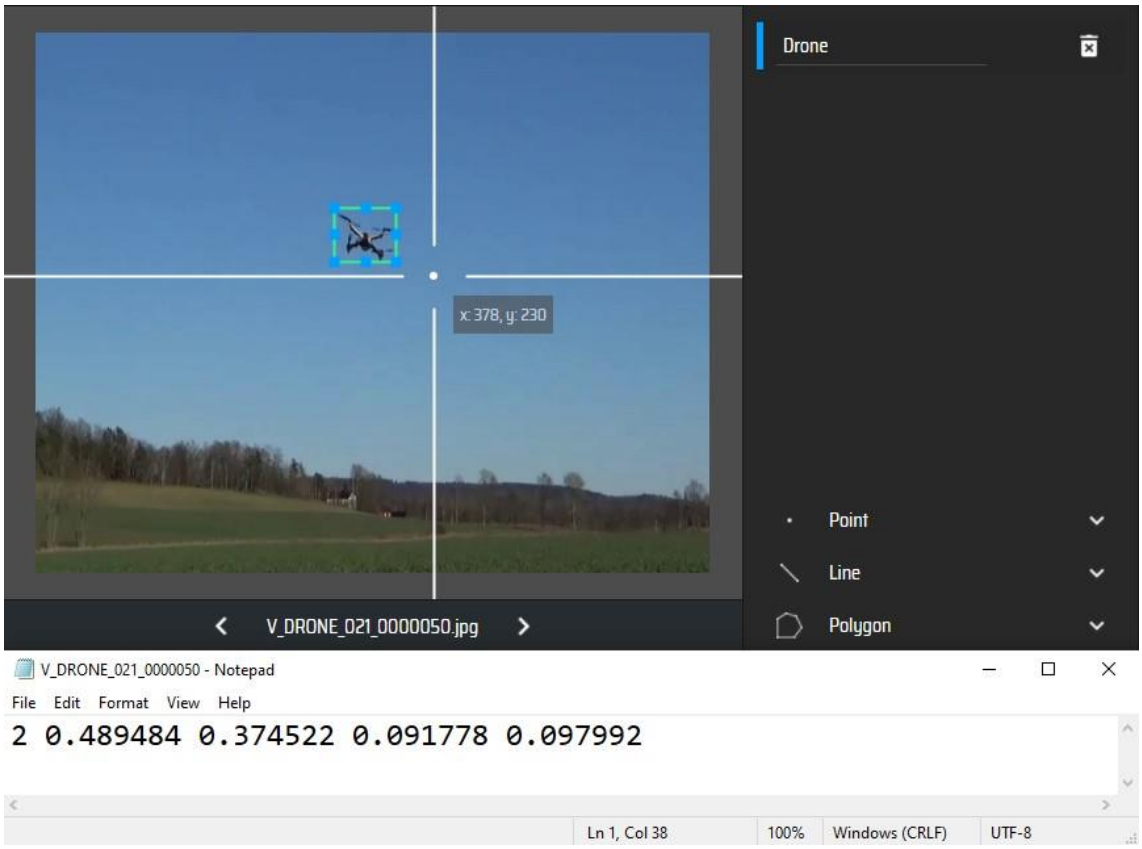
Formát *YOLO* byl vybrán z důvodu následného trénování modelu *YOLOv5*, který využívá tento stejnojmenný formát. Výhodou je také jednoduchá čitelnost a přehlednost zápisu.

Samotný formát popisku obrázku *YOLO* je následující. Každý objekt je zapsán na každém řádku zvlášť. Počet popsaných řádků odpovídá počtu orámovaných objektů na obrázku. Každý řádek obsahuje pět čísel oddělených mezerou. První číslo je označení třídy objektu. Další čtyři čísla vytváří rámeček objektu. Obě souřadnice jsou normovány (0, 1). Druhé a třetí číslo udává střed orámované oblasti (souřadnice *x* a *y*). Čtvrté a páté číslo definuje rozměry orámované oblasti (šířka a výška). Jméno obrázku a popisku je stejný, liší se pouze přípona souboru (*txt/jpg*). Graficky je proces tvorby popisku znázorněn na obrázku 3-4.



Obr. 3-4 Ukázka popisku obrázku ve formátu YOLO

Pro samotný proces tvorby popisků se nejlépe osvědčila webová stránka *Make Sense* [17]. Hlavní výhodou této stránky je jednoduchost a intuitivní ovládání. Navíc jsou data v bezpečí, protože nejsou nikam odesílány. Na obrázku 3-5 je zobrazen proces tvorby popisku v programu *Make Sense* a výsledný exportovaný popisek ve formátu *YOLO*.



Obr. 3-5 Ukázka tvorby popisku v prostředí Make Sense [16] [17]

3.7.5 Stanovení parametrů obrázků

Na základě charakteru obrázků jsou stanoveny parametry, které jsou použity pro optimální výběr snímků pro následné reverzní vyhledávání. Stanovit parametry ručně je časově náročné, navíc se opakuje v každé iteraci vyhledávání nových obrázků. Z tohoto důvodu bylo zvoleno použití automaticky generovaných parametrů. Pro přibližné rozřazení obrázků bude pro tuto úlohu stačit generování parametrů na základě popisku a

analýzy histogramu obrázku.

Stanovení parametrů bylo provedeno pomocí vlastního skriptu v jazyce *python compute_parameters_of_images.py*. Počítání parametrů je rozděleno do dvou funkcí *get_parameters_based_on_label()* a *get_parameters_of_weather_and_complexity()*.

První funkce vychází z informací získaných z popisku. Konkrétní parametry jsou: třída, velikost oblasti a počet objektů. Velikost oblasti je vztažena k objektu, který zaujímá největší plochu z označených objektů. Plocha je spočtena jednoduše jako obsah obdélníku (čtverce), který ohraničuje objekt. Na základě velikosti plochy jsou rozlišeny tři stavy, malá, střední a velká oblast. Podle počtu objektů je stanoven výstupní stav, s jedním objektem nebo s více objekty. Tato funkce je univerzální.

Druhá funkce využívá histogram obrázků k odhadu počasí a členitosti snímku. Konkrétně je pro odhad počasí využito zbarvení oblohy. Z histogramu jsou spočteny průměrné hodnoty jednotlivých barevných složek. Použitá barevná složka a dělící kritérium bylo stanoveno na základě praktického zkoušení. Vybrána byla modrá složka s dělící hodnotou 107. Aby bylo dosaženo výraznějšího kontrastu, je histogram vytvořen pouze z horní poloviny snímku, na které se zpravidla vyskytuje obloha. Rozlišeny jsou dva stavy, slunečno a proměnlivo.

Obr. 3-6 Odhad počasí na základě histogramu obrázku [16]

Pro odhad členitosti je použit histogram z celého obrázku. Z histogramu je následně vypočtena směrodatná odchylka. Například směrodatná odchylka na snímku modré oblohy je menší než na snímku s krajinou. Výstupem je opět rozdělení do dvou stavů, lze je nazvat například nečlenitý a členitý snímek. Dělící kritérium na základě směrodatné odchylky bylo nastaveno na hodnotu 31. Opět je proces znázorněn graficky, ukázka je na obrázku 3-7.

Obr. 3-7 Odhad členitosti na základě histogramu obrázku [16]

Funkce *get_parameters_of_weather_and_complexity()* už není plně univerzální, ale minimálně parametr členitosti lze použít u široké palety úloh.

Parametry jsou uloženy v názvu textového souboru. Pro každý obrázek je vytvořen samostatný prázdný textový soubor, název je poskládán ze dvou částí, první část tvoří název obrázku, druhá část obsahuje pět číslic, které nesou informaci o parametrech.

Obr. 3-8 Ukázka a popis formátu parametrů [16]

3.7.6 Výběr vhodných obrázků pro reverzní vyhledávání

Posledním krokem jedné iterace je výběr vhodných obrázků pro reverzní vyhledávání. Poté lze zahájit novou iteraci rozšiřování datasetu. Vybrán může být jakýkoliv obrázek, který byl získán v předchozím iteračním algoritmu, avšak nesmí se jednat o snímek, který už byl pro reverzní vyhledávání použit. Výběr snímků je proveden zcela automaticky pomocí skriptů v *pythonu*. Ručně jsou zadávány pouze požadavky na dataset a požadovaný počet vybraných snímků.

Prvním možným postupem je výběr obrázků pomocí požadavku na složení datasetu. V této časti bude pracováno s vlastním skriptem *analyze_diversity__create_new_iter.py*. Prvním krokem je nastavení procentuálního zastoupení hodnot parametrů. Dále je nutné nastavit požadovaný počet vybraných obrázků. Podle požadovaného počtu obrázků je spočten počet vybraných snímků pro každou kombinaci parametrů. Počty pro každou kombinaci jsou následně zaokrouhleny na celé číslo. Z tohoto důvodu se může konečný

počet vybraných prvků mírně lišit od počtu požadovaného. Navíc dosavadní dataset nemusí obsahovat obrázky se všemi požadovanými kombinacemi parametrů. V takovém případě jsou tyto požadavky přeskočeny. Příklad nastavení požadavků na složení výběru je zobrazen na obrázku 3-9.

```
# TYPE SELECTION ############################################################
# class (airplane, bird, drone, helicopter, insect)
class_name = [20, 20, 20, 20, 20]
# complexity of image (50% low; 50% high) # parameter [%]
complexity = [50, 50]
# area of object (40% low, 40% medium, 20% high) [%]
area = [40, 40, 20]
# weather in image (50% sunny, 50% not_sunny)
weather = [50, 50]
# number of images with multiple objects (just for print, only information)
multiple_object_min = 10
```

Obr. 3-9 Požadavek na složení výběru obrázků

Skript lze spustit ve třech módech. První mód je vhodný pro analýzu a zkoušení nastavení. Není vytvořen výběr, pouze je vypsáno, jaké obrázky by byly vybrány a kolik procent z požadovaného počtu obrázků je k dispozici. Druhý mód už slouží k vytvoření výběru a zkopírování obrázku do zvolené složky. Třetí mód je vylepšením předchozího módu, ale využívá navíc analýzu dosavadního datasetu. Jak získat tuto analýzu je popsáno v následujícím odstavci.

K analýze dosavadního datasetu slouží skript *analyze_merge_dataset_diversity.py*. Vstupem jsou parametry jednotlivých obrázků datasetu a požadované složení datasetu. Stejně jako ve skriptu *analyze_diversity__create_new_iter.py*, je požadované složení zadáno, jak je zobrazeno na obrázku 3-9. Výstupem je seznam kombinací parametrů, které jsou v dosavadním datasetu zastoupeny dostatečně. Tento seznam je uložen v textové podobě. Následně lze použít skript *analyze_diversity__create_new_iter.py* ve třetím módu. Výběr obrázků probíhá stejně jako ve druhém módu, ale jsou vynechány kombinace parametrů ze seznamu (dostatečně zastoupené kombinace).

3.8 Získaný dataset

Popsaný algoritmus získávání dat fungoval bez problému, kromě získávání třídy hmyz. Důvodem je specifické zadání úlohy. Primárním cílem kamery jsou větší létající objekty, především drony. Pokud je zachycen na záběru hmyz, předpokladem je poměrně nekvalitní snímek. Požadavkem je proto získání obrázků, na kterých je hmyz na hranici rozpoznatelnosti, například rozostřené snímky. Na vyhledávačích obrázků jsou ale upřednostněny především velmi kvalitní obrázky. Takto kvalitních obrázků není možné v reálných podmínkách dosáhnout. Třída hmyz byla proto dohledána především manuálně a jsou zde použity i statické snímky hmyzu z datasetu [18].

Pomocí dříve popsaného algoritmu a skriptů bylo realizováno pět iterací vyhledávání obrázků. V tabulce 3-1 jsou počty objektů vyšší, protože obrázek může obsahovat více objektů.

Tabulka 3-1 Zastoupení tříd v získaném datasetu

Třída	Letadlo	Pták	Dron	Helikoptéra	Hmyz
Počet obrázků	649	494	399	477	122
Počet objektů	776	869	439	717	403

Pro ilustraci jsou na obrázku 3-10 zobrazeny příklady.

Obr. 3-10 Ukázka obrázků z vlastního datasetu [19] [20] [21] [22] [23] [24]

Lehce podhodnoceny jsou třídy dron a hmyz. Menší počet dronů může být vykompenzován pomocí datasetu 285, protože zde je naopak třída dron nejpočetnější. Doplnění třídy hmyzu by bylo vhodné provést až po vybrání a nastavení snímací aparatury. Cílem je získat co nejlepší výsledky u všech ostatních tříd, poté je vhodné určit jaký hmyz je možné aparaturou zachytit a jestli je možné tuto třídu klasifikovat. Podle získaných poznatků bude databáze třídy hmyz rozšířena, nebo neuvažována.

Opět se zde nabízí použití skriptu *analyze_merge_dataset_diversity.py*. Stačí pouze označit složku s parametry obrázků a program spustit. Výsledky analýzy datasetu jsou zobrazeny na obrázku 3-11. Zastoupení tříd je vypočteno na základě počtu obrázků.

```
class distribution in dataset (Airplane) [%]: 30.25
class distribution in dataset (Bird) [%]: 23.09
class distribution in dataset (Drone) [%]: 18.65
class distribution in dataset (Helicopter) [%]: 22.3
class distribution in dataset (Insect) [%]: 5.7

area distribution in dataset (small) [%]: 34.6
area distribution in dataset (medium) [%]: 28.94
area distribution in dataset (large) [%]: 36.47

complexity distribution in dataset (small) [%]: 56.47
complexity distribution in dataset (big) [%]: 43.53

weather distribution in dataset (sunny) [%]: 41.47
weather distribution in dataset (not_sunny) [%]: 58.53
```

Obr. 3-11 Výstup skriptu pro analýzu diverzity získaného datasetu

4. ROZBOR ŘEŠENÍ ÚLOHY

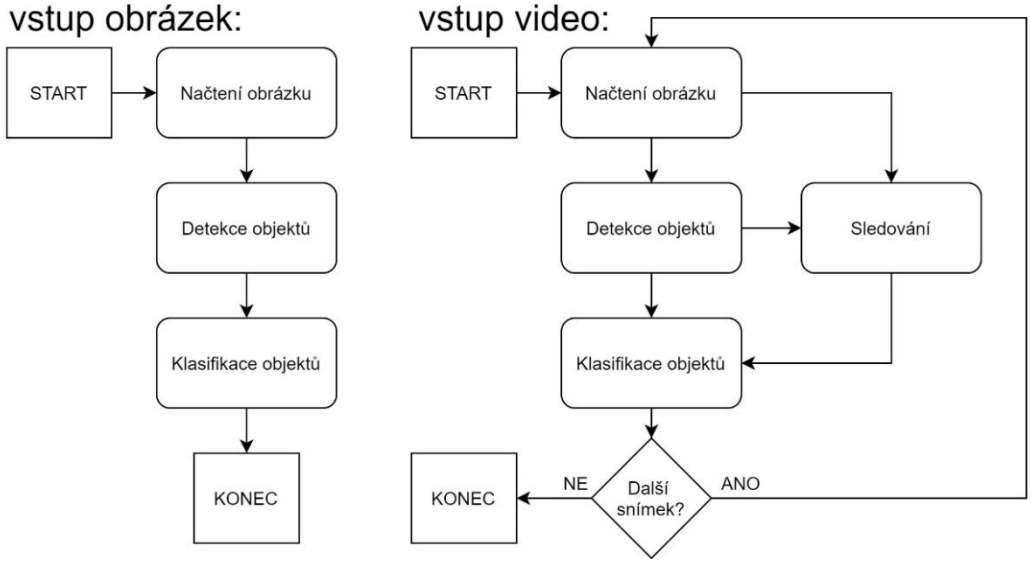
První možné řešení vychází z teoretického úvodu, konkrétně z kapitoly 2. Byly popsány nejpoužívanější modely neuronových sítí pro detekci a klasifikaci objektů z jednoho snímku komplexní scény. Z těchto modelů bude vybrán jeden jako zástupce. Předpokladem je použití předtrénovaného modelu a následného dotrénování na vlastním datasetu.

4.1 Rozdělení úlohy na detekci, trasování a klasifikaci

Alternativou k použití modelů pro detekci a klasifikaci z jednoho snímku je rozdělení úlohy do jednotlivých kroků. Výsledkem bude modulární program. Program musí obsahovat modul detekce a klasifikace. Možností je přidat trasování detekovaných objektů z minulých snímků. Výsledným dílem je modulární program, který funguje podobně jako modely pro detekci a klasifikaci z jednoho snímku. Ovšem na rozdíl od těchto modelů, program využívá jako vstup sérii snímků (video).

Mezi hlavní výhody patří možnost průběžné kontroly jednotlivých části programu a modularita řešení. Návrh může být prováděn postupně, lze měřit výkonost jednotlivých částí programu a podle výsledků tyto části modifikovat. Například kontrola detekce může být provedena nezávisle na klasifikačním modulu.

Použití klasických metod pro detekci je vhodné především pro úlohy se specifickým zadáním a obecně tam, kde jsou známy nebo lze stanovit omezující podmínky. Na obrázku 4-1 je znázorněn obecný přístup, pro detekci a klasifikaci objektů, který je použit v této práci.



Obr. 4-1 Obecný algoritmus detekce a klasifikace

Co se týče tohoto zadání, pokud budeme uvažovat kompozici se statickou kamerou, lze úlohu zjednodušit.

V prvním kroku je vykonáván proces detekce (například pomocí odečtení pozadí). Vytipované oblasti jsou následně prověřeny pomocí *CNN* pro klasifikaci létajících objektů. Pokud je na oblasti klasifikován objekt, začíná trasování objektu. S určenou periodou, nebo s aktualizací trasování, je opět provedena klasifikace pomocí *CNN*. Třída objektu je stanovena na základě určeného počtu posledních klasifikačních výsledků. Nabízí se možnost jednoduché výstupní logiky, která upřednostní třídu, která byla klasifikována nejčastěji.

Návrh a implementace tohoto přístupu bude dál vedena pod názvem „Modulární program“ a jedná se o alternativu k prvnímu přístupu „Model pro detekci a klasifikaci z jednoho snímku“.

4.2 Software pro řešení úlohy

Pro oblast počítačového vidění, strojového učení a umělé inteligence je k dispozici široká řada knihoven. Co se týče programovacích jazyků, nejčastěji se v současné době využívá *python*, použít však lze i jiné jazyky, například *MATLAB* nebo *C++*. V této kapitole se bude práce zabývat pouze *pythonem* a jeho *open source* knihovnami.

4.2.1 OpenCV

Prvním zástupcem knihovny počítačového vidění a učení je *OpenCV*. Důvodem vzniku byla snaha poskytnout společnou infrastrukturu pro aplikace počítačového vidění. Knihovna obsahuje více než dva a půl tisíce optimalizovaných algoritmů. Například lze uvést algoritmy detekování a rozpoznání obličeje, identifikace objektů, sledování pohybu objektů a získání 3D modelů objektů. [25]

4.2.2 PyTorch

Knihovna byla primárně vyvinuta laboratoří AI pro výzkum firmy *Facebook*. Obsahuje flexibilní a modulární *framework* pro práci s pokročilými metodami umělé inteligence. Zároveň je obsah knihovny optimalizován pro použití *GPU*. Mezi hlavní domény této knihovny patří práce s neuronovými sítěmi. [26]

4.2.3 TensorFlow

Za vývojem knihovny stojí *Google Brain Team*. Stejně jako *PyTorch* je i tato knihovna kromě jiného hlavně zaměřena na neuronové sítě a obě knihovny cílí na uživatele s podobnými požadavky. [27]

4.2.4 Google Colaboratory

Hlavní výhodou použití *Google Colab* je získání výkonné výpočetní platformy v okně internetového prohlížeče. Služba je dostupná i v bezplatné verzi a pro základní použití stačí pouze *Google* účet a stabilní internetové připojení. Zjednodušeně napsáno se jedná o spustitelný dokument s jednoduchým sdílením souborů prostřednictvím *Google Drive*.

Kód a komentáře jsou zapisovány do *Jupiter notebooku* v jazyce *python*. Výhodou je dostupnost základních knihoven bez nutnosti dodatečné instalace. Dále je možné kód spouštět postupně po sekcích programu. Jako úložiště lze použít *Google Drive*.

5. IMPLEMENTACE MODELU PRO DETEKCI A KLASIFIKACI Z JEDNOHO SNÍMKU

Prvním krokem je určení konkrétního modelu pro detekci a klasifikaci z jednoho snímku. Bylo rozhodnuto o vybrání jednoho z modelů popsaných v kapitole 2. K přihlédnutí k úloze, bude vybrán jeden z výpočetně méně náročných modelů. Nabízí se použití modelu *YOLO* a *SSD*. Nakonec byl vybrán model *YOLO*. Hlavním důvodem je nová verze *YOLOv5*, která kromě vylepšení výsledků přináší uživatelsky přívětivé trénování a používání modelu.

Cílem je natrénovat model *YOLOv5* pro detekci a klasifikaci objektů. Použit bude vlastní dataset, jehož tvorba a skladba je popsána v kapitole 6. Následně bude model podroben zkoušce na datasetu 285. Zásadní je zde rozhodnutí datasety nesloučit. Důvodem je ověření robustnosti modelu. Vlastní dataset je vytvořen pomocí reverzního vyhledávání a dataset 285 se skládá převážně z videí pořízených speciálně pro tvorbu datasetu létajících objektů. Videa datasetu 285 můžou sloužit jako ukázka reálného nasazení modelu v praxi, ale nejsou příliš pestrá. Proto budou sloužit pouze jako testovací data.

5.1 Trénovací dataset

Trénovací data tvoří pouze vlastní dataset. Z důvodu vyrovnání zastoupení tříd nebyla použita všechna data. Při trénování na celém datasetu bylo dosaženo horších výsledků.

Tabulka 5-1 Zastoupení obrázků v trénovacím datasetu

Třída	Letadlo	Pták	Dron	Helikoptéra	Hmyz
Počet obrázků	399	399	399	399	122
Počet objektů	485	726	403	586	403

5.2 Učení Modelu YOLOv5

V této podkapitole je využita předchozí příprava. Výše zmíněné části jsou použity pro naučení Modelu *YOLOv5*. Vlastní dataset byl rozdělen na trénovací (70 %) a validační data (30 %). Rozdělení bylo provedeno pomocí vlastního skriptu v jazyce python *shuffle_split_dataset.py*. Možností skriptu je zvolit požadovaný poměr rozdělení do jednotlivých složek.

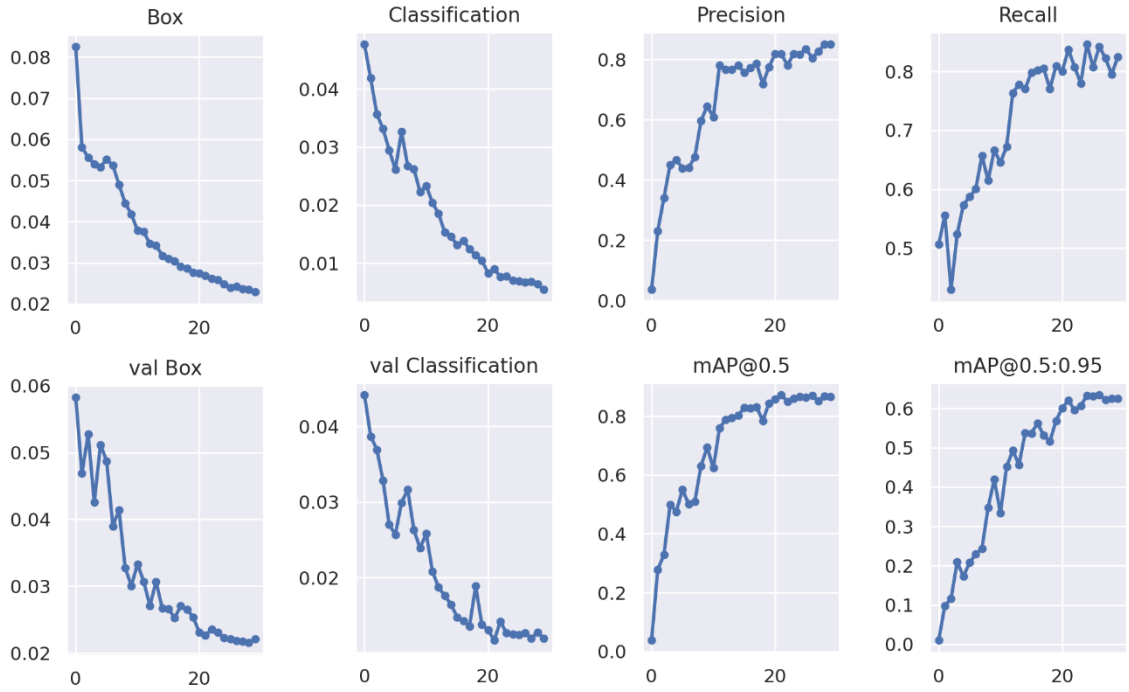
Model *YOLOv5* je možné použít v různých modifikacích architektury. Na výběr je ze čtyř základních variant (*s*, *m*, *l* a *x*). Pro trénování byla konkrétně vybrána verze *YOLOv5l (large)*, která obsahuje $47 \cdot 10^6$ parametrů sítě. Byl použit předtrénovaný model. Trénování probíhalo po dávkách (16 obrázků v každé). Celkem bylo provedeno třicet trénovacích cyklů. Ukázka vstupních dat je zobrazena na obrázku 5-1.

Obr. 5-1 Ukázka obrázků z trénovacího datasetu s popisky [28] [16]

Model *YOLOv5* používá pro dosažení robustního modelu augmentaci obrázků. Ukázka augmentovaných dat je zobrazena na obrázku 5-2. Protože model *YOLO* v prvním kroku změní rozlišení obrázku na požadovanou úroveň, musí model zvládat i zdeformované obrázky. Dále může být nový obrázek poskládán z více původních snímků.

Obr. 5-2 Ukázka augmentovaných obrázků trénovacího datasetu s popisky [28] [16]

Model *YOLOv5* také ukládá informace z trénování. Průběh ztrátových funkcí je zobrazen na obrázku 5-3. První řádek se váže k trénovacím datům, druhý řádek reprezentuje validační data. Lze vyčíst, že počet iteračních cyklů je dostatečný, protože ztrátové funkce u validačních dat přestaly klesat. Další informace z trénování lze nalézt v příloze ve složce *YOLO/trénování*



Obr. 5-3 Statistiky z průběhu trénování modelu YOLOv5

6. NÁVRH A IMPLEMENTACE MODULÁRNÍHO PROGRAMU

V této kapitole bude navržen modulární program pro detekci a klasifikaci létajících objektů. Postup vychází z kapitoly 4, kde byl proved rozbor možných řešení.

Je nutné ošetřit speciální stav, kdy je sledovaná oblast vyhodnocena jako „pozadí“. Důvodem může být přehodnocení sledované oblasti, selhání algoritmu trasování nebo se může objekt dostat mimo záběr. V tomto případě je trasování ukončeno. Je nutné najít rovnováhu mezi předčasným ukončením trasování a dlouhým trasováním oblasti bez objektu. V prvním příkladě může být létající objekt dočasně zakryt jiným předmětem. Následně předpokládáme novou detekci, jakmile se objekt znova objeví, ale už je s ním nakládáno jako s novým objektem. Historie minulých detekcí byla ztracena. V druhém případě může nastat trasování pozadí, které pouze výpočetně zatěžuje systém.

Při detekci může docházet k vytipování již sledovaných objektů. Tento stav je nutné rozpoznat a nezahájit nové trasování oblasti. Při neodhalení tohoto stavu dochází k chybné vícenásobné detekci objektu a zároveň roste výpočetní náročnost (klesá počet zpracovaných snímků za časový údaj). Pokud je vícenásobné trasování jednoho objektu zahájeno, je nutné pomocí kontrolních mechanismů redundantní trasování identifikovat a ukončit.

6.1 Detekce

Prvním krokem algoritmu je detekce objektu. V této části lze uplatnit celou řadu různých přístupů. V prvé řadě lze metody rozdělit podle vstupu algoritmu. Vstup je zpravidla snímek, nebo série snímků (video). Pro řešení této úlohy je předpokládán jako vstup série snímků.

Bude uvažováno použití statické kamery, popřípadě kamery s otáčením do stran, která funguje po většinu času jako statická kamera. Tato situace nabízí možnost detekce pomocí odečtení předchozího snímku, popřípadě průměru z předchozích snímků. Většina létajících objektů se pohybuje dostatečnou rychlostí, aby rozdíl mezi jednotlivými snímky (popřípadě průměru snímků) byl dostatečný pro detekci. Výhodou popsaného přístupu je možnost detekce objektů, které zaujímají malou plochu snímků (vzdálené a malé objekty). Další výhodou je jednoduchost. Nevýhodou je naopak nutnost filtrovat falešnou detekci, která je způsobena pohyby kamery nebo změnou kompozice pozadí, například pohybem větví ve větru.

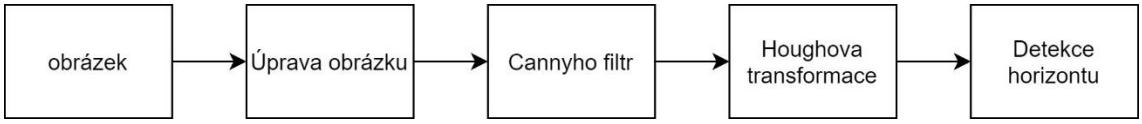
Metody založené na odečtu pozadí se vyznačují dobrým poměrem mezi účinností a výpočetní náročností. Pro tuto konkrétní úlohu je vhodné doplnit algoritmus detekce o funkci hledání horizontu obrázku, protože pod horizontem jsou falešné detekce nejčastější a létající objekty se pod horizontem obrázku vyskytují ve snížené míře. Dále

je možné odhadnout nežádoucí pohyb kamery, který se vyznačuje velkou změnou obrazu mezi snímky. Při pohybu kamery je vhodné detekci přerušit.

V následujících podkapitolách budou popsány nástroje, které jsou použity pro detekci. Na tomto místě je nutné vyzdvihnout knihovnu *OpenCV* [25]. Tato knihovna obsahuje širokou paletu funkcí od práce s obrázky až po aplikace počítačového vidění.

6.1.1 Hledání horizontu

Jako inspirace pro tvorbu funkce hledání horizontu posloužila práce [29]. Principem je použití *Cannyho filtru* a *Houghovy transformace*. Obě tyto funkce jsou dostupné v knihovně *OpenCV*. Postup je naznačen na obrázku 6-1.



Obr. 6-1 Detekce horizontu [29]

Konkrétně se jedná o funkce *Canny()*, která slouží jako detektor hran, aplikuje se na šedotónový a výstupem je binární obrázek. Druhou funkcí je *HoughLine()*. Jako vstup slouží výstup *Cannyho filtru*. Výstupem jsou přímky, které dělí obrázek v místech hran. Pomocí popsaných funkcí a výstupní logiky lze sestrojit jednoduchý detektor horizontu.

6.1.2 Odečtení pozadí

V této části budou popsány algoritmy pro odečítání pozadí, které jsou implementovány v knihovně *OpenCV*.

Mezi nejpoužívanější metody odečítání pozadí se řadí metoda *Gaussovy směsi* (*Mixture of Gaussians*). Zkratka metody je *MOG*. Za zmínku stojí především druhá verze implementace této metody *createBackgroundSubtractorMOG2()*. Princip spočívá ve sledování pixelů. Pokud hodnoty pixelu zůstávají déle stabilní, roste pravděpodobnost, že se jedná o pixel pozadí.

Další přístup využívá *Bayesovské* statistiky. Výsledná metoda je implementována v knihovně *OpenCV*, jméno funkce je *createBackgroundSubtractorGMG()*.

6.2 Trasování

Tento bod není pro celkový algoritmus nezbytný. Avšak trasováním objektů je uchována informace ze snímků minulých. Například výstupní hodnotu *CNN* pro klasifikaci objektů. Metody sledování lze rozdělit do dvou kategorií. V první kategorii jsou metody, které dokážou nalézt objekt i na dalším snímku. Obecně se tyto metody nazývají *tracker*. Do druhé kategorie patří metody, které pracují s detekcí z různých snímků a dokážou sloučit dohromady detekce stejného objektu. Například na základě polohy. V této práci byly vyzkoušeny přístupy z obou kategorií.

6.2.1 Na základě polohy

Jedná se o velmi jednoduchou a výpočetně nenáročnou metodu. Nevýhodou je závislost na detekci objektů a nižší spolehlivost u většího počtu objektů na jednom snímku. Konkrétně byla použita implementace z článku [30]. Metoda je založena na principu porovnávání *Euklidovy* vzdálenosti současných detekcí a detekcí z předchozího snímku, pokud nepřekročí nastavenou mez, je detekci přiřazeno stejné identifikační číslo.

6.2.2 Pomocí trackeru

Opět je možné využít nabídky z knihovny *OpenCV*, která obsahuje implementaci několika *trackerů*.

Prvním krokem je inicializace *trackeru*, která se provádí pomocí snímku s vyznačeným objektem. Vyznačení je provedeno pomocí obdélníkového orámování oblasti. V dalších iteracích je vstupem funkce *trackeru* pouze snímek. Výsledkem je aktualizace obdélníkového ohraničení podle pohybu sledovaného objektu. Předběžně byly vytipovány *trackery*, které úspěšně trasovaly letadlo na videu z datasetu 285.

Obr. 6-2 Zkouška trackerů na jednoduché scéně (CSRT, MIL, MOOSE)

6.3 Klasifikace

Pro klasifikaci lze použít libovolnou metodu. Předpoklad je však použití *CNN*. V této práci byl vybrán model architektury *ResNet*. Důvodem je použití přímého propojení mezi vybranými vrstvami modelu. Důsledkem této modifikace je zvýšena pravděpodobnost úspěšného natrénování modelu. Konkrétně byl zvolen model *ResNet50*. Jedná se o kompromis mezi výkonem a výpočetní náročnosti modelu.

6.4 Popis programu s trasováním na základě polohy

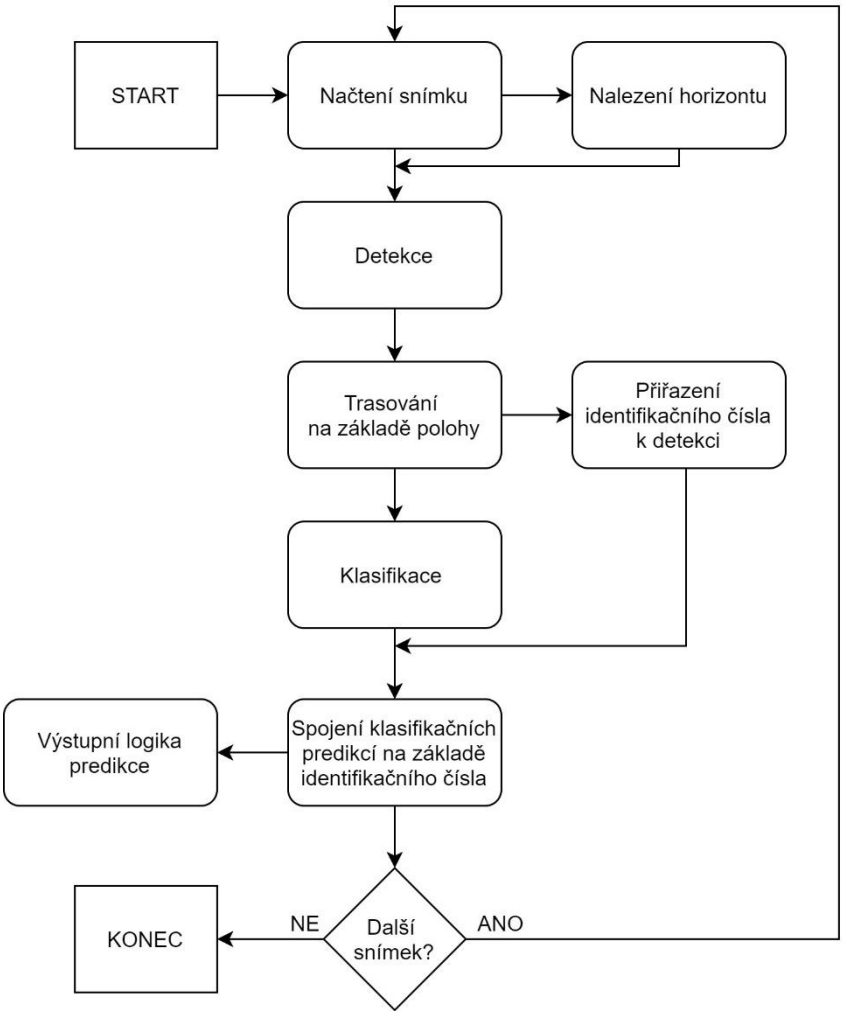
Program s trasováním na základě polohy je poměrně jednoduchý. Zjednodušeně si lze představit pouze funkci detekce a následné klasifikace. Ostatní části pouze zlepšují výkonost a algoritmus je funkční i bez nich. Podrobně je program popsán pomocí blokového diagramu na obrázku 6-3.

Detekce byla realizovaná pomocí funkce *createBackgroundSubtractorMOG2()*. Princip této funkce a celkový princip detekce při použití statické kamery je popsán v podkapitole 6.1. Zde je nastaven maximální počet oblastí, které je možné zpracovat

během jednoho cyklu. Důvodem je výpočetní náročnost *CNN* pro klasifikaci oblastí. Algoritmus byl rovněž doplněn o hledání horizontu. Detekce zaznamenané pod horizontem jsou ignorovány. Konkrétní postup je opět rozveden v podkapitole 6.1.

Na základě polohy jsou přiřazeny jednotlivým detekcím unikátní identifikační čísla. Následně jsou detekované oblasti ořezány a vyhodnoceny pomocí *CNN*, v tomto případě pomocí modelu *ResNet*. Uvažovaná je také třída pozadí, což má za následek snížení počtu falešných detekcí, ale na druhou stranu také odečítá správné detekce, pokud chybně klasifikuje objekt jako pozadí. S jistým nadhledem je možné zařadit model *CNN* nejen do části klasifikace, ale i do bloku detekce.

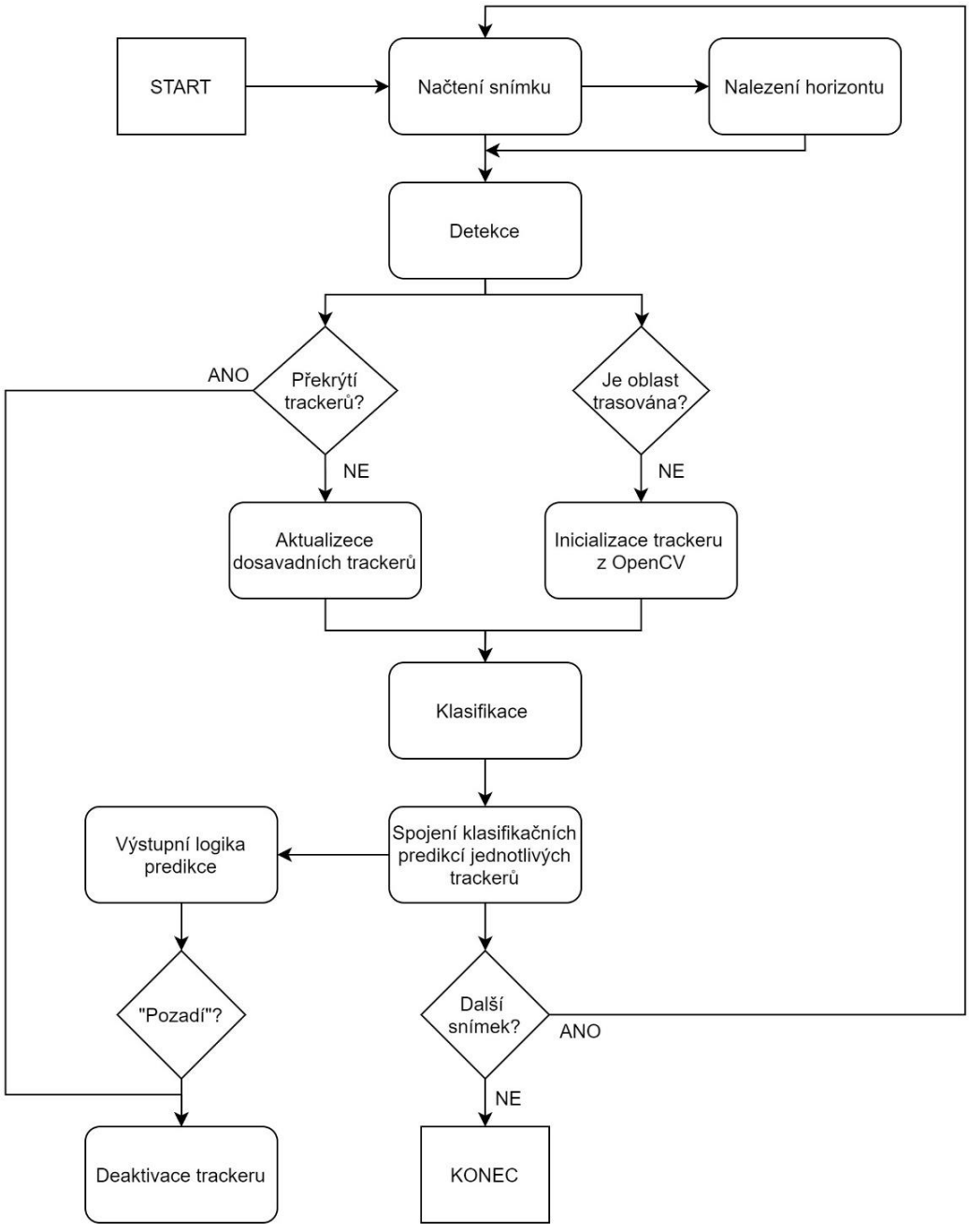
Dalším krokem je spojení současných a předchozích klasifikačních předpovědí. Ke každému identifikačnímu číslu jsou přiřazeny aktuální predikce. Pomocí výstupní logiky je stanovena konečná predikce třídy. Vykonávání programu buď končí, nebo je načten další snímek a cyklus se opakuje.



Obr. 6-3 Blokový diagram programu s trasováním na základě polohy

6.5 Popis programu s trasováním pomocí trackeru

V této podkapitole bude věnována pozornost pouze rozdílům s předchozím programem. Důvodem je podobnost obou programů a je zbytečné opakovat popis ostatních částí algoritmů podruhé. Algoritmus je vysvětlen pomocí blokového diagramu na obrázku 6-4.



Obr. 6-4 Blokový diagram programu s trasováním pomocí trackeru

Při použití *trackerů* z knihovny *OpenCV* je nutné ošetřit více stavů. Vlastnost *trackerů* je taková, že dokážou trasovat vždy pouze jednu oblast. Proto je nutné přiřadit ke každému trasování samostatnou instanci *trackeru*. Z tohoto důvodu musí být

v programu ošetřena režie mezi těmito instancemi a dále je nutné provádět proces inicializace a deaktivace. Opět je vhodné stanovit maximální počet současně běžících instancí *trackeru* z důvodu výpočetní náročnosti.

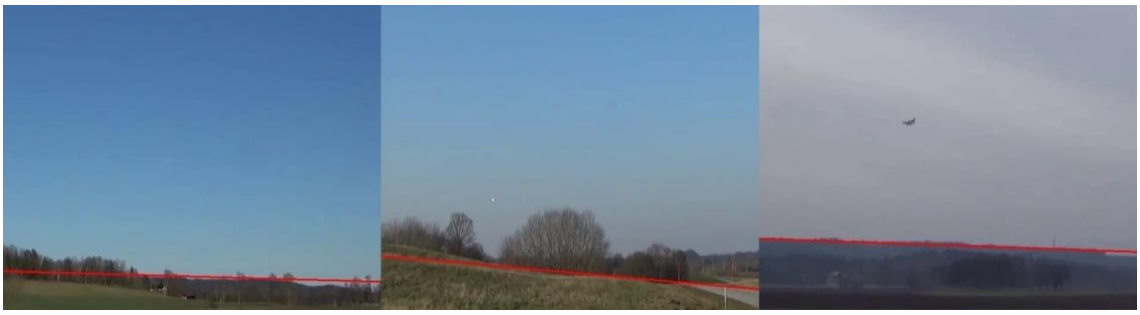
Časová délka výpočtu aktualizace *trackerů* a predikce pomocí modelu *ResNet* závisí na konkrétních vybraných modelech, ale doba výpočtu je řádově stejná. Proto je vhodné provádět tyto výpočty paralelně (rozdělení do paralelních vláken). Výhodou je zrychlení programu. Dále je možné pomocí tohoto přístupu ošetřit inicializaci *trackeru*, která trvá řádově déle než aktualizace. Program nečeká na dokončení inicializace a pokračuje dál ve vykonávání. Jakmile je inicializace dokončena, aktualizace *trackeru* je už dále prováděna v každé smyčce algoritmu. Nevýhodou paralelismu je provádění klasifikace se zpožděním o jeden průchod smyčkou.

6.6 Implementace programu s trasováním na základě polohy

V této podkapitole bude probrán proces implementace programu. Postupně budou využity funkce a modely popsané v této kapitole. Vše je nutné vyzkoušet na testovacích videích a zařídit propojení jednotlivých částí, jak je vysvětleno v blokovém diagramu. Rovněž je nutné zjistit kolik snímků je možné zpracovat za časový úsek, například za sekundu (*fps*). Důvodem je předpokládané nasazení programu v reálném čase. Nízký počet zpracovaných snímků za jednotku času může způsobit zhoršení výsledků programu.

6.6.1 Detektor horizontu

Původně nebyl detektor horizontu použit. Přesto výsledky detekce nebyly špatné. Ale ve zvýšené míře docházelo k falešné detekci v oblasti horizontu a pod ním. Proto byl dodatečně detektor horizontu přidán. Detekce pod vyznačeným horizontem jsou ignorovány.



Obr. 6-5 Ukázka výstupu funkce odhadu horizontu

Funkce detektoru horizontu funguje na principu popsaném v podkapitole 6.1.1. Zde už je potřeba popsat pouze výstupní logiku aplikovanou na rozdělení snímku pomocí přímek (výstup *Houghovy transformace*).

Detektor horizontu je volán při každém načtení nového snímku. Po aplikování *Houghouvy transformace* je pracováno pouze s nejvýznamnější přímkou. Horizont je předpokládán přibližně vodorovný, proto je kontrolován úhel přímky. Je tolerována

odchylka 9º od horizontální polohy. Pokud se přímka nenachází v této toleranci, není započítaná. Výstupem funkce je průměrná hodnota z desíti posledních platných přímek. Proto se na vstupu funkce vkládá i historie detektoru. Jedná se o velmi primitivní výstupní logiku. Algoritmus chybuje při složitějších scénách. Určitě je možné dosáhnout lepších výsledků, ale pro většinu situací je řešení dostatečné.

6.6.2 Detekce objektů

Provedení detekce objektů je postaveno na funkci *createBackgroundSubtractorMOG2()* z knihovny *OpenCV*, která vykonává odečtení pozadí. Funkce je určena speciálně pro použití se statickou kamerou. Výstupem jsou vyznačené oblasti, které nejsou statické. Dále jsou použity funkce *threshold()* (pro odstranění detekcí stínů) a *findContours()* (pro ohraničení detekcí. Využity jsou pouze detekce s plochou větší než osm pixelů. Zdroj této části kódu je článek [30].

Nastavení osmi pixelů je optimální pro detekci, avšak objekty o této velikosti nejsou příliš kvalitně klasifikovány, občas prostě není možné třídu objektu určit. Pro lepší klasifikační výsledky je možné predikovat třídu pouze pro oblasti s větší plochou. Konkrétní hodnota plochy bude stanovena později na základě výstupu z testování programu.

Následně byla ještě přidána část, která indikuje pohyb kamery. Toho je docíleno pomocí počítání celkového počtu predikcí a zaznamenání souhrnné plochy těchto predikcí. Pohyb kamery je indikován při celkové ploše detekcí nad dva tisíce pixelů nebo překročení počtu patnácti detekcí překračující plochu osmi pixelů. Přiklad s pohyblivou kamerou je zobrazen na obrázku 6-6.

Obr. 6-6 Ukázka detekce při pohybu kamery

6.6.3 Trasování na základě polohy

Použitý kód je k nalezení v článku [30]. Byla pouze změněna hodnota, která udává maximální vzdálenost mezi jednotlivými detekcemi, kdy jsou ještě tyto detekce připisovány stejnému zdroji. Momentálně je použita hodnota padesát pixelů. Zde je předpoklad nastavení podle konkrétní použité kamery a průměrného počtu zpracovaných snímků za sekundu.

6.6.4 Klasifikace

Predikce je provedena pomocí modelu *ResNet50*. Původně trénování proběhlo pouze

na vlastním datasetu, ale nebylo dosaženo použitelných výsledků. Proto byl trénovací dataset doplněn o část původně testovacích dat. Jednalo se o přibližně třetinu testovacích videí. Z každé třídy zastoupené v testovacím datasetu bylo přemístěno třicet videí do trénovacího datasetu. Videa v testovacím datasetu jsou často velmi podobná. Aby bylo možné ověřit robustnost modelu, nelze použít příliš podobné snímky na trénování a testování. Zpravidla se v testovacím datasetu vyskytují podobná videa hned za sebou. Proto byl vybrán souvislý blok třiceti videí od každé třídy. Z každého videa byly získány zhruba tři snímky. Ke každé třídě (kromě třídy hmyz) bylo přidáno zhruba sto snímků k trénování.

Dataset bylo nutné upravit pro učení klasifikačního modelu. Původně byly objekty součástí komplexního obrázku, v doplňujícím dokumentu byly vypsány třída a pozice objektů (formát *YOLO*). Pro trénování klasifikační *CNN* bylo potřeba tyto objekty ze snímků vystřihnout a vytvořit tak dataset pouze s objekty bez pozadí. Konkrétně model *ResNet* používá rozlišení snímku 224 × 224, všechny výřezy jsou proto přepočítány na tento rozměr. Dále byla přidána třída pozadí, která má za cíl odhalit falešnou detekci. Vystřižení objektů a náhodného pozadí snímku, bylo provedena pomocí vlastního skriptu *creating_images_for_CNN_classification.py*. Je vhodné ještě doplnit, že lze nastavit provedení ořezu objektu. Například lze přidat větší okraj kolem objektu nebo provést řez přesně podle orámování z popisku. Přidání okraje navíc je výhodné pro trénování, protože v rámci augmentace lze vybrat pokaždé trochu jinou část. Konkrétně byla přidána polovina *šířky/výšky* popisku na každou stranu výřezu.

Tabulka 6-1 Zastoupení tříd v trénovacím datasetu pro klasifikační model

Třída	Letadlo	Pták	Dron	Helikoptéra	Hmyz	Pozadí
Počet obrázků	882	994	534	842	403	2060

Trénování bylo provedeno pomocí *colab* notebooku [31]. Prostředí *Google Colaboratory* je popsáno v podkapitole 4.1.4. Kód bylo potřeba pouze mírně upravit, například zvolit požadovaný *ResNet* model a nastavit šest výstupních tříd.

Byl použit předtrénovaný model a následně bylo provedeno celkem čtyřicet iteračních cyklů. Trénování probíhalo po dávkách (8 obrázků v každé). Pro optimalizaci byl použit stochastický gradient. *Learning rate* (parametr učení) byl nastaven na hodnotu *0.001*. Navíc se tento parametr zmenší na desetinu z původní hodnoty každých sedm iterací. Během trénování byly vyzkoušeny různé nastavení náhodného ořezu trénovacích obrázků. Nakonec byl vybrán postup s přepočítáním vstupního snímku na rozměry 340 × 340 a následný náhodný výběr oblasti o rozměrech 224 × 224.

V tomto odstavci bude popsáno použití modelu *ResNet50* v modulárním programu. Nejprve je podle detekcí vytvořeno pole s ořezanými oblastmi, které byly označeny pomocí detekce. Je možné nastavit, jestli bude ořezání provedeno přesně podle souřadnic detekce, nebo bude přidáno i blízké okolí, případně kolik. Nakonec je nutné obrázek

přepočítat na rozměry 224 × 224. Poté je už celé pole posláno k predikci do klasifikačního modelu *ResNet50*.

6.6.5 Výstupní predikce

Predikce jsou ukládány do dvourozměrného pole, první index pole je identifikační číslo detekovaného objektu a druhý index označuje pořadí predikce tohoto objektu. Nová predikce je vždy připojena na konec. Výstupní predikce je stanovena pro všechny aktuální detekce. Logika výstupní predikce je jednoduchá. Použito je osm posledních *CNN* predikcí pro daný objekt a výstupem je třída s nejvyšším zastoupením.

6.6.6 Důsledky rychlosti zpracování snímků

Při zpracování v reálném čase může docházet k přeskakování snímků, které program nestihl zpracovat. Z této situace vyvstává otázka, jak moc vynechané snímky ovlivní celkový výkon programu. V tomto případě má vynechání snímků přímý vliv na detekci a trasování. Nepřímo je ovlivněna výstupní predikce třídy objektu, protože je klasifikováno méně obrázků, což zvyšuje nejistotu výstupní predikce.

Pro objektivní zkoušení programu na testovacích videích, je nutné přeskakovat snímky, tak aby video běželo původní rychlostí. Počet přeskočených snímků lze vypočítat na základě potřebného času k zpracování snímku a snímkové frekvenci videa (*frame rate*).

Výpočet kolik snímků je potřeba přeskočit:

$$n = max\,\{round[(t_2 - t_1) \cdot frame\ rate] - 1,\ 0\}, \tag{6.1}$$

kde *n* je počet snímků k přeskočení, t_2 je aktuální čas, t_1 je čas před načtením minulého snímku a *frame rate* je snímková frekvence videa.

Zaokrouhlení je nutné z důvodu proveditelnosti. Funkce *max* ošetřuje případy, kdy je výpočet rychlejší než snímková frekvence videa.

6.7 Implementace programu s trasováním pomocí trackeru

Pro začátek je použitý kód algoritmu z předchozí podkapitoly 6.6. Rozdílem je pouze použitý prostředek pro trasování. Rozdíly při použití tohoto programu jsou již popsány v kapitole 6.5. Zbývá popsat samotný blok trasování.

6.7.1 Trasování pomocí trackerů

Obecné informace o *trackerech* z knihovny *OpenCV* jsou uvedeny v podkapitole 6.2.2. Vyzkoušeny byly všechny dostupné modely. Nejlépe fungoval *MIL tracker* [32].

6.8 Srovnání navržených programů

Ze srovnání programů vychází lépe program s trasováním na základě polohy. Důvodem jsou neuspokojivé výsledky trasování pomocí *trackerů* z knihovny *OpenCV*. Trasování

selhávalo především u menších objektů a docházelo ke zpoždění ohraničení objektu. Nejspíše je to způsobeno tím, že jsou *trackery* optimalizovány pro trasování větších objektů. Avšak v této úloze se často pracuje s velmi malou plochou detekovaných objektů. Navíc je řešení s trasováním na základě polohy výpočetně méně náročné. Proto je vhodné dále pracovat pouze s jedním programem, kterým je program s trasováním na základě polohy, popsaný v podkapitolách 6.4 a 6.6.

Na obrázku 6-7 lze vidět kompletní grafický výstup programu. Hlavní okno zobrazuje detekci. První číslice detekce je identifikační číslo. Druhá číslice odkazuje na výřez, který je zobrazen v menším okně.

U vedlejších oken je červeně napsáno číslo aktuální predikce a žlutě je výstupní hodnota predikce určená na základě výstupní logiky. (třídy: 0 - letadlo, 1 - pták).

Obr. 6-7 Výstup programu s trasováním na základě polohy

Video-ukázka běhu modulárního programu je v příloze *Modulární_program/ukázka.*

7. TESTOVÁNÍ

Postupně bylo provedeno testování modelu *YOLOv5* a modulárního programu. Nakonec byly výsledky srovnány na stejných datech.

7.1 Testovací modelu YOLOv5

Pro testování byl použit dataset 285. Byly použity všechny videa. Data bylo nutné upravit pro trénink modelu *YOLOv5*.

7.1.1 Získání snímků z videa

Je nutné provést rozdělení videa na snímky. Zároveň je nutné tyto snímky uložit s požadovaným jménem, aby název korespondoval se souborem s popiskem objektů. Pro tuto úlohu byl vytvořen *python* skript *video_to_images.py*.

Je třeba rozhodnout, jestli budou použity všechny snímky z videa. Mezi jednotlivými snímky nedochází k výrazné změně obrazu, proto byly vytvořeny různé datasety. Rozsah je od jednoho snímku na video (celkově 285 obrázků) až po použití všech snímků (celkově 89 233 obrázků). Pro testování byl zvolen kompromis, kdy byl použit každý padesátý obrázek videa (celkové 1140 obrázků).

Obr. 7-1 Ukázka snímků z testovacího datasetu [16]

7.1.2 Vytvoření popisků v YOLO formátu pro testování

Popis a orámování objektů bylo vytvořeno autorem datasetu. Anotace byla provedena v programu *MATLAB*. Ke každému videu z datasetu přísluší jeden soubor s příponou *.mat*. Soubor s popisky je ve formátu *groundTruth*.

Převedení formátu popisku bylo provedeno pomocí vlastního *MATLAB* skriptu *label_yolo.m*, který převedl soubor ve formátu *groundTruth* na *YOLO* formát. Tento formát popisuje jednotlivé obrázky [33].

7.1.3 Výsledky

V této části byl vyzkoušen natrénovaný model *YOLOv5 large* na testovacích snímcích. Predikce modelu se skládá ze souřadnic a názvu třídy. Navíc je zobrazena také důvěra samotného modelu ve správnost predikce. Zobrazeny jsou pouze predikce, které překračují požadovaný práh důvěry modelu. Pomocí nastavení tohoto parametru lze výrazně měnit výsledky testování. Důvěra je u jednotlivých predikcí označena i na obrázku 7-2.

Pro vyhodnocení testování byl využit vlastní skript *test_YOLO_on_images.py*. Vstupem jsou popisky obrázků a textový výstup z testování.

V tabulkách 7-1 a 7-2 jsou zobrazeny výsledky testování. Při testování byl nastaven práh pro detekci 0,6 (*conf-0.6*). Výsledkem je nízký počet detekcí, ale detekované objekty jsou predikovány s poměrně vysokou úspěšností.

Pro vyhodnocení je použitá čtyřpolní tabulka, predikce může nabývat čtyř stavů, *True positive* (*TP*) označuje správné označení objektu, popřípadě správné určení třídy objektu, *False positive* (*FP*) znamená falešné označení objektu nebo chybné určení třídy (vztahuje se k třídě, která je predikovaná), *False negative* (*FN*) reprezentuje situace, kdy není objekt detekován, pro klasifikaci se jedná o chybu určení třídy (vztahuje se ke skutečné třídě objektů), *True negative* (*TN*) má smysl vyhodnocovat pouze u klasifikace, jedná se o správné neoznačení třídy (třída daného objektu je jiná).

Pro klasifikaci jsou dále vypočteny hodnoty pro vyhodnocení testování.

Výpočet přesnosti:

$$přesnost = \frac{TP}{TP+FP} \tag{7.1}$$

Výpočet citlivosti:

$$citlivost = \frac{TP}{TP+FN} \tag{7.2}$$

Výpočet skóre F_1:

$$F_1 = \frac{TP}{TP+\frac{1}{2}(FP+FN)} \tag{7.3}$$

Zpravidla jsou výsledky převedeny na procenta, důvodem je větší přehlednost.

Tabulka 7-1 Statistika detekce na testovacích datech (conf-0,6)

	Skutečný počet	Správně označených (TP)	Falešné označení (FP)	Duplicitní označení (FP)
Letadlo	236	72	20	0
Pták	242	53	9	2
Dron	468	157	1	0
Helikoptéra	252	35	2	0

Tabulka 7-2 Statistika klasifikace na testovacích datech (conf-0,6)

	TP	FP	FN	TN	přesnost	citlivost	F_1[%]
Letadlo	68	51	4	195	57,1	94,4	71,2
Pták	53	10	1	254	84,1	98,1	90,6
Dron	104	1	53	160	99,0	66,2	79,4
Helikoptéra	29	2	6	281	93,5	82,9	87,9

V tabulkách 7-3 a 7-4 jsou zobrazeny výsledky testování při nastavení důvěry na hodnotu 0,4 (*conf-0,4*). Docíleno je lepší detekce objektů, ale jejich predikce už není tak úspěšná.

Tabulka 7-3 Statistika detekce na testovacích datech (conf-0,4)

	Skutečný počet	Správně označených (TP)	Falešné označení (FP)	Duplicitní označení (FP)
Letadlo	236	105	22	10
Pták	242	90	12	2
Dron	468	266	3	18
Helikoptéra	252	204	3	10

Tabulka 7-4 Statistika klasifikace na testovacích datech (conf-0,4)

	TP	FP	FN	TN	přesnost	citlivost	F_1[%]
Letadlo	94	139	16	436	40,3	85,5	54,8
Pták	85	42	6	552	66,9	93,4	78,0
Dron	142	27	133	383	84,0	51,6	64.0
Helikoptéra	151	4	58	472	97,4	72,2	83,0

Pokud srovnáme výsledky testování s různým nastavením (*conf*), tak v obou případech bylo dosaženo zajímavých výsledků. Konečné nastavení záleží na konkrétním použití modelu. Lepší detekce je docíleno s nižší potřebnou důvěrou modelu. Naopak klasifikace je kvalitnější s vyšší hodnotou důvěry.
Výstup z testování modelu lze najít v příloze ve složce *YOLO/testování*. Na obrázku 7-2 je zobrazeno několik snímků z této složky. (*conf-0.6*)

Obr. 7-2 Ukázka detekce modelu YOLOv5 na testovacím datasetu [16]

7.2 Testování modulárního programu

V této kapitole je popsáno vyzkoušení modulárního programu na testovacích datech. Vzhledem k modularitě algoritmu, jsou vyhodnoceny jednotlivé části samostatně a poté je hodnocen i celkový algoritmus. Následně může být posouzeno, na které části algoritmu se vyplatí soustředit, při budoucím vylepšování programu. Rovněž lze odhadnout potenciál celkového programu.

Zároveň je při testování programu simulována skutečná rychlost videí. To znamená, že v důsledku vyšší snímkové frekvence videa než frekvence zpracování, byl „přeskočen“ adekvátní počet snímků. Detailněji je proces popsán v podkapitole 6.6.6. Při použití modulárního programu je zpracováno několik snímků během sekundy. Což může odpovídat například zpracování každého pátého obrázku testovacího videa. Záleží především na počtu detekcí, které jsou klasifikovány pomocí *CNN*. V tomto případě jsou klasifikovány na jednom snímku maximálně tři detekce. Dále samozřejmě záleží na použitém hardwaru. Testování bylo provedeno na notebooku vybaveném procesorem *Intel Core i5-8265U* a grafickou kartou *NVIDIA GeForce MX150*.

I když je prováděno vyhodnocení detekce samostatně, stále je uvažovaná rychlost zpracování jako při použití celého programu.

7.2.1 Testování detekce

Pro testování detekce je použit dataset 285. Jedná se o stejný dataset jako v případě testování modelu *YOLOv5*. Avšak testování je provedeno pouze na videích se statickou kamerou.

Testování proběhlo pomocí vlastního skriptu *test_algorithm_on_test_videos.py*. Tento skript je použit i při testování celého algoritmu. Vstupem jsou testovací videa s popisky ve formátu *YOLO* (pro přehlednost jsou popisky celého videa uloženy v jednom textovém souboru).

Vyhodnocovány jsou pouze snímky, které byly modelem zpracovány. Predikce a popisek jsou poté vyhodnoceny samostatně pro každý obrázek. Pro vyhodnocení detekce jako úspěšné, je potřeba alespoň částečné překrytí popisku a predikce. Dále se započítávají falešné detekce a redundantní označení při úspěšné detekci (duplicitní označení). Ještě je nutné poznamenat, že falešné označení je počítáno pouze v případě, že výstupní třída není vyhodnocena jako *pozadí*.

Občas se při detekci vyskytne i duplicitní označení. Důvodem je nastavení minimální plochy pro detekci, nastavena je hodnota osmi pixelů. Nastavení je optimální pro detekci objektů s malou plochou. U větších objektů hrozí duplicitní označení.

Tabulka 7-5 Statistika detekce na testovacích statických videích

	Skutečný počet	Správně označených (TP)	Falešné označení (FP)	Duplicitní označení (FP)
Letadlo	11248	2271	214	34
Pták	8867	849	106	14
Dron	27560	8574	63	295
Helikoptéra	2251	384	8	24

7.2.2 Testování trasování

Zvolená metoda je natolik primitivní, že není potřeba výsledky samostatně testovat. Ovšem je zajímavé sledovat, jaký vliv bude mít nastavení trasování na celkovou výkonost programu. Jediným nastavitelným parametrem je vzdálenost mezi detekcemi, při které jsou ještě detekce (na současném a předchozím snímku) přiřazeny ke stejnému identifikačnímu číslu. Pokud je číslo nastaveno na nulu, lze simulovat chování algoritmu bez trasování. Výstupní třída predikce bude odpovídat klasifikační předpovědi z aktuálního snímku. Poté lze srovnat výsledky s trasováním a „bez trasování“. Na základě těchto výsledků lze rozhodnout o přínosu modulu trasování. Výsledky jsou ukázány v podkapitole 7.2.4, kde je program testován jako celek.

7.2.3 Testování klasifikace

Testování modelu *ResNet50* proběhlo na snímcích z videí, které nebyly použité při trénování. Z jednotlivých snímků videa byly vyříznuty objekty podle popisků. Jedná se o stejný proces jako u tvorby trénovacího datasetu. Vystřižení obrázků bylo provedeno

přesně podle popisků. Aby byl obrázek pro klasifikaci blíže skutečným podmínkám, je vystřižená oblast nejprve přepočítána na rozměr 260 × 260 a poté je z tohoto obrázku provedeno náhodné vybrání o rozměrech 224 × 224. Konkrétně byl použit vlastní skript *test_CNN_on_cropped_images_from_videos.py*.

Tabulka 7-6 Statistika klasifikace modelu ResNet50 na testovacích datech

	TP	FP	FN	TN	přesnost	citlivost	F_1[%]
Letadlo	574	307	150	1744	65,2	79,2	71,5
Pták	513	311	151	1800	62,3	77,3	69,0
Dron	462	178	351	1784	72,2	56,8	63,6
Helikoptéra	334	2	240	2199	99,4	58,2	73,4

7.2.4 Testování celého programu

Pro otestování fungování celého programu jsou použity videa z datasetu 285. Samozřejmě je použita pouze část, která nebyla využívaná při trénování. Zároveň jsou vybrány pouze videa se statickou kamerou. Avšak je zde pár výjimek, například pohyb kamery na konci videa.

Provést celkovou kontrolu algoritmu je nutné, protože při posílání detekcí ke klasifikaci lze nastavit ořezání oblasti, což má podstatný vliv na výkonost klasifikace. Rozšíření ořezu bylo prováděno podle funkce *max(15, šířka detekce*) na každé straně ve směru osy *x*. Obdobně byl ořez rozšířen i ve směru osy *y*. Důvodem je ošetření situace, kdy je detekována pouze část objektu.

Byly vyzkoušeny dvě varianty nastavení klasifikace. První variantou bylo predikování třídy u všech detekcí. Druhou variantou bylo provedení odhadu třídy pouze u detekcí s významnějšími rozměry, konkrétně s plochu detekce větší než 24 pixelů. Překvapivě bylo dosaženo přibližně stejné úspěšnosti klasifikace u obou nastavení. Proto budou uvedeny pouze výsledky s predikcí u všech detekcí (nad 8 pixelů). Rovněž byl vyzkoušen vliv vypnutí trasování objektů. Výsledkem je, že výstupní predikce závisí pouze na jediné klasifikaci.

V tabulce 7-7 je potvrzeno správné fungování celého programu. Klasifikační výsledky jsou přibližně stejné jako při samostatném testování modelu *ResNet50*. Zároveň lze vyčíst zlepšení klasifikačních výsledků při trasování objektů.

Tabulka 7-7 Porovnání klasifikace s trasováním a bez trasování

	TP	FP	FN	TN	přesnost	citlivost	F_1[%]
S trasováním	2337	627	797	8775	78,8	74,6	76,6
Bez trasování	2216	686	884	8614	76,4	71,5	73,8

Jelikož není získáno dostatečné množství detekcí pro všechny třídy testovacího datasetu, je klasifikace vyhodnocena pouze pro všechny třídy dohromady. Vyhodnocení klasifikace pro každou třídu zvlášť bylo provedeno v podkapitole 7.2.3.

Už není potřeba hodnotit detekci, tento úkon byl proveden v podkapitole 7.2.1. Rozdíl byl pouze v použití většího testovacího datasetu. Důvodem byla možnost využít i videa, která byla zařazena do trénovacího datasetu modelu *ResNet50*.

7.3 Srovnání použitých přístupů

Pro možnost porovnání detekce a klasifikace modelu *YOLOv5* a modulárního programu, je nutné pracovat se stejným testovacím datasetem. Pro vyhodnocení modelu *YOLOv5* byl použit celý dataset 285, ale u modulárního programu byly pro vyhodnocení detekce použity pouze videa se statickou kamerou. U kontroly klasifikace modelu *ResNet50* nebyly použity videa, které byly součástí trénovacího datasetu. V této kapitole bude model *YOLOv5* vyhodnocen na stejných datech.

7.3.1 Statistika detekce

Nejprve byly porovnány výsledky detekce na statických videích. Pro jednoduchost je vyhodnocení provedeno pro všechny třídy dohromady.

Tabulka 7-8 Statistika detekce na statických testovacích datech

	Počet	Správně označených (TP)	Falešné označení (FP)
YOLOv5 (conf-0,6)	819	209 (25,5 %)	26 (3,17 %)
YOLOv5 (conf-0,4)	819	403 (49,2 %)	33 (4,03 %)
Modulární program	49926	12078 (24,2 %)	758 (1,52 %)

7.3.2 Statistika klasifikace

V případě vyhodnocení klasifikace detekovaných oblastí, je nutné použít pouze data, která nebyla použita při trénování modelu *ResNet*. Zároveň stále platí vybrání pouze statických videí. Opět jsou vyhodnoceny všechny třídy dohromady. Především z důvodu nižšího počtu detekcí na testovacím datasetu.

Tabulka 7-9 Statistika klasifikace na testovacích datech

	TP	FP	FN	TN	přesnost	citlivost	F_1[%]
YOLOv5 (conf-0,6)	40	13	13	137	75,5	75,5	75,5
YOLOv5 (conf-0,4)	73	42	42	303	63,5	63,5	63,5
Modulární program	2337	627	797	8775	78,8	74,6	76,6

7.3.3 Zhodnocení výsledků

Před komentářem výsledků je vhodné dodat, že testovací dataset obsahoval poměrně složité snímky pro detekci a klasifikaci. Na několika snímcích je obtížné klasifikovat objekt i pro člověka. Na druhou stranu jsou obsaženy i scény s poměrně jednoduchou detekcí a klasifikací. Vyřazena byla množina videí, které nejsou pořízeny statickou kamerou. Tyto videa byly většinou z kategorie „jednodušších“, například scény s přiblížením na letící objekt. Proto pro testování v této kapitole zůstaly především snímky s „pixelově“ méně významnými objekty. Což se odráží v poměrně nízké úspěšnosti detekce u obou přístupů. Ukázku výřezů objektů z testovacích videí lze vidět na obrázku 7.3.

Obr. 7-3 Ukázka objektů z testovacího datasetu [16]

Pro modulární program je rovněž předpoklad lepších výsledků při praktickém použití. Důvodem je testování na krátkých videích. Funkce odečtu pozadí začíná pracovat spolehlivě až po určitém množství snímků.

Při hodnocení klasifikace vychází nepatrně lépe modulární program s použitým modelem *ResNet50*. Je však nutné dodat, že pro trénování modelu *ResNet50* bylo použité větší množství dat. Aby byl model dostatečně robustní, bylo potřeba přidat snímky z původně testovacích videí. Pro lepší výsledky je nutné rozšířit stávající dataset. První variantou je praktikovat reverzní vyhledávání obrázků i pro jiné webové vyhledávače než *Yandex*. Proces může být zopakován podle popsaného postupu v kapitole 3. Jednodušší variantou je však získání obrázků z obecných datasetů. Lze použít i kompromis, kdy některé třídy můžou být získány pomocí reverzního vyhledávání a zbytek nalezen ve vytvořených datasetech.

8. NÁVRH HARDWARU

Při návrhu hardwaru lze popsat dva přístupy. První možností je použití libovolného počítače nebo notebooku. Jako kamera lze použít i lepší webkamera. Je nutné počítat s tím, že grafický výkon a kvalita kamery ovlivňuje výkon modelu/programu. Výhodou tohoto řešení je možnost vyzkoušení obou přístupu bez nutnosti pořizování dalšího hardwaru.

Druhý přístup je vhodný pro kontinuální běh modelu/programu. Cílem návrhu bude vytvoření kompaktní sestavy, která bude mít nízký odběr energie (do 15 W). Z tohoto důvodu bude provedena rešerše pouze pro jednodeskové počítače. Bude uvažováno napájení ze sítě pomocí adaptéru. Podle konkrétních požadavků lze použít i bateriové napájení, popřípadě doplnit celý přípravek o solární panel. Pro venkovní umístění přípravku je nutné rovněž počítat s nutností ochrany proti atmosférickým vlivům. Opět může být použita libovolná kamera, ale je dobré zvážit pořízení kamery odpovídající specifickým požadavkům.

8.1 Rešerše jednodeskového počítače

Při rešerši byly uvažovány dvě kategorie jednodeskových počítačů. První kategorií jsou univerzální jednodeskové počítače. Avšak vhodnější jsou jednodeskové počítače se specializací na strojové učení, které jsou zahrnuty ve druhé kategorii.

8.1.1 Univerzální jednodeskový počítač

Tyto počítače nejsou specializované na konkrétní činnost. Lze je použít pro širokou paletu aplikací od náhrady klasického počítače až po různé úlohy strojového učení. Nabídka v této kategorii je velmi rozmanitá. Nejznámějším a nejpoužívanějším zástupcem jsou jednodeskové počítače *Raspberry Pi*. Z tohoto důvodu bude tento počítač zahrnut v tabulce 8-1.

8.1.2 Jednodeskový počítač se specializací

Tyto jednodeskové počítače jsou navrženy přímo pro aplikace umělé inteligence, strojového učení a počítačového vidění. Proto je kladen důraz na výpočetní výkon u výše zmíněných aplikací, důsledkem je zvýšený grafický výkon.

První možností je použít rozšíření jednodeskového počítače pomocí *USB* adaptéru, například *Google Coral* nebo *Intel Neural Stick 2*.

Druhou možností je použití jednodeskového počítače určeného pro tento druh úloh**,** například výrobek *Jetson Nano* od firmy *NVIDIA*. Tento výrobek nabízí lepší poměr *cena/výkon*, něž použití univerzálního jednodeskového počítače s výše zmíněnými *USB* adaptéry [34]. Proto bude *Jetson Nano* použit pro srovnání v tabulce 8-1.

Tabulka 8-1 Srovnání vybraných jednodeskových počítačů

<table>
<tr><td></td><td>Raspberry Pi 4 Model B - 4GB</td><td>Raspberry Pi 4 Model B - 8GB</td><td>Jetson Nano DK</td><td>Jetson Nano 2GB DK</td></tr>
<tr><td>CPU</td><td colspan="2">1.5 GHz quad-core ARM A72</td><td colspan="2">1.43 GHz quad-core ARM A57</td></tr>
<tr><td>GPU</td><td colspan="2"></td><td colspan="2">128-core NVIDIA Maxwell</td></tr>
<tr><td>Paměť</td><td>4 GB LPDDR4</td><td>8 GB LPDDR4</td><td>4 GB LPDDR4</td><td>2 GB LPDDR4</td></tr>
<tr><td>Napájení</td><td colspan="4">5 V, 3 A</td></tr>
<tr><td rowspan="5">Konektivita</td><td colspan="2">2.4GHz a 5GHz Wi-fi</td><td></td><td></td></tr>
<tr><td colspan="2">Bluetooth 5.0</td><td></td><td></td></tr>
<tr><td colspan="4">Gigabit Ethernet</td></tr>
<tr><td colspan="2">2 x USB 2.0 konektor</td><td></td><td>2 x USB 2.0 konektor</td></tr>
<tr><td colspan="2">2 x USB 3.0 konektor</td><td>4 x USB 3.0 konektor</td><td>1 x USB 3.0 konektor</td></tr>
<tr><td></td><td colspan="4">MIPI CSI-2</td></tr>
<tr><td>Maloobchodní cena (s DPH)</td><td>1499 Kč</td><td>2149 Kč</td><td>2976 Kč</td><td>1692 Kč</td></tr>
</table>

Mezi jednotlivými modely *Raspberry Pi* je rozdíl pouze mezi paměťmi RAM. U modelů *Jetson Nano* je navíc mírný rozdíl v konektivitě a nižší model může připojit pouze jednu kameru přes rozhraní *MIPI CSI-2*.

8.1.3 Zhodnocení

Modely *Raspberry Pi* disponují lepší celkovou výbavu než modely *Jetson Nano*, navíc jsou levnější. Avšak díky přidanému *GPU* jsou modely *Jetson Nano* o třídu lepší v aplikacích počítačového vidění a jiných aplikacích, které využijí vyššího grafického výkonu.

Vzhledem k výpočetní náročnosti popsaných přístupů v kapitolách 5 a 6, je vhodné vybrat nejvýkonnější možnost z tabulky (*Jetson Nano Developer Kit)*. Pro tento jednodeskový počítač lze použít popsané metody detekce a klasifikace bez větších úprav. Při použití modelu *YOLOv5 large*, lze přibližně předpokládat zpracování jednoho snímku (640 × 512) za sekundu, což může být stále dostatečná rychlost pro podstatnou část aplikací. Pro použití modulárního programu s využitím modelu *ResNet50* budou nutné úpravy. Je těžké posoudit v jakém rozsahu by bylo nutné algoritmus zjednodušit. Nabízí se ale hned několik možností, provádět klasifikaci pouze v určitém intervalu, klasifikovat pouze jednu oblast nebo natrénovat méně komplikovaný *CNN* model, například *ResNet34*.

8.2 Rešerše kamery

Jak bylo řečeno v úvodu kapitoly, je možné použít i kvalitní webkameru. Testování navržených přístupů proběhlo na obrázcích/videích s rozlišením 640 × 512 pixelů. Lze doporučit pracovat se stejným nebo větším rozlišením. U modelu *YOLOv5* je provedena detekce a klasifikace v jednom kroku. Zvětšení rozlišení může zlepšit detekci i klasifikaci. U modulárního programu bude mít rozlišení větší vliv na klasifikaci. Důležitější, než samotné rozlišení je kvalita obrázků. Rozostření a rozmazání snímku negativně ovlivňují zejména klasifikaci objektů.

Při výběru kamery k jednodeskovému počítači, je vhodné uvažovat pouze kamery s připojením přes rozhraní *MIPI CSI-2*. Levnějším řešením je volba kamery bez možnosti nastavení objektivu. Například *IMX219-77 8Mpx*. Druhým řešením je zvolit senzor s možností výměny objektivu. Jedná se o nákladnější možnost, ale výhodou je vyšší kvalita snímků a zvolení optimálního objektivu pro konkrétní požadavky. Jako příklad lze uvést kameru *IMX477 12.3Mpx*. Tato kamera je k dostání i ve verzi se softwarově ovládaným ostřením.

(a) IMX219-77 8Mpx (b) IMX477 12.3Mpx (c) IMX477 12.3Mpx

Obr. 8-1 Ukázka možných kamer pro přípravek Jetson Nano [35] [36] [37]

V tabulce 8-2. jsou zobrazeny parametry kamer použitých na obrázku 8-1. Pro většinu aplikací lze doporučit kameru *IMX219-77 8Mpx*. Důvodem je dobrý poměr *cena/výkon*.

Tabulka 8-2 Parametry vybraných kamer

	IMX219-77 8Mpx	IMX477 12.3Mpx	
		Výměnný objektiv	Softwarové ostření
Rozlišení senzoru	3280 × 2464	4056 × 3040	
Ohnisková vzdálenost	2.96 mm	Podle objektivu	3.9 mm
Zorné pole	77º	Podle objektivu	75º
Maloobchodní cena (s DPH)	18.99 $ (~401 Kč)	59.99 $ (~1267 Kč)	79.99 $ (~1690 Kč)

ZÁVĚR

V teoretická části práce byly vysvětleny základní principy neuronových sítí. Pozornost byla věnována konvoluční neuronové síti, která se používá při práci s vizuálním vstupem. Byly představeny příklady klasifikačních neuronových sítí.

Teoretická část pokračovala popisem modelů, které jsou schopny detekovat a klasifikovat objekty na snímku s komplexní scénou. Tyto modely jsou založeny na konvolučních neuronových sítích. V této oblasti bylo dosaženo velkého pokroku během několika posledních let. Například model *R-CNN* byl vytvořen v roce 2014. Od té doby prošel tento model několika vylepšeními. Navíc byly vyvinuty další modely, které nabídly vyšší rychlost zpracování obrazu, například modely *SSD* a *YOLO*.

V kapitole zabývající se tvorbou vlastního datasetu byl vysvětlen obecný postup pro získání datasetu pomocí reverzního vyhledávání obrázků. Poté byl obecný postup aplikován na zadanou úlohu a byl vytvořen vlastní dataset obrázků. Pro řešení byly napsány různé skripty v jazyce *python*, které zjednodušily a zrychlily proces tvorby datasetu. Výhodou je možnost použití popsaného postupu pro tvorby různých datasetů. Rovněž podstatná část vytvořených skriptů je opětovně využitelná u dalších úloh. Vytvořený dataset je dostatečně pestrý, ale pro budoucí použití by bylo vhodné dataset ještě rozšířit. Například použitím různých webových vyhledávačů obrázků nebo zkombinováním s jiným už vytvořeným datasetem.

Z popsaných modelů, které detekují a klasifikují na základě jednoho snímku, byl vybrán jako zástupce model *YOLOv5*. Důvodem výběru je kombinace vysoké výkonosti modelu a implementace v knihovně *pytorch*, což umožňuje jednoduché použití. Konkrétně byl vybrán model *YOLOv5 large*. Trénován probíhalo pouze na vlastních datech, které byly získané v kapitole 3 (Tvorba vlastního datasetu).

Dalším bodem byl návrh a implementace modulárního programu. Cílem bylo vytvořit program, který bude využívat informace z předcházejících snímků. Detekce je založena na principu odečítání pozadí a použití statické kamery. Pro trasování byly původně uvažovány *trackery* z knihovny *OpenCV*, ale výsledky nebyly uspokojivé. Bylo proto vybráno jednodušší řešení s trasováním na základě polohy. Pro klasifikaci detekovaných oblastí byl použit model *ResNet50*.

V další kapitole byly obě navržené řešení otestovány. Pro testování byl použit dataset z práce [16]. Nejprve byl otestován model *YOLOv5*, bylo použito přes tisíc testovacích obrázků. Dále byl otestován modulární program. Na rozdíl od testování modelu *YOLOv5*, byly použity celá videa, protože princip detekce je založen na odečítání pozadí, které potřebuje dynamickou scénu. Bylo nutné použít pouze videa se statickou kamerou. Dále byly pro kontrolu klasifikace modelu *ResNet50* vyřazeny videa, která sloužila pro trénování klasifikátoru.

Aby bylo možné porovnat výsledky testování modelu *YOLOv5* a modulárního programu, byla provedena statistika pouze u společných testovacích dat.

Ze srovnání výsledků a komentáře v kapitole 7 lze usuzovat, že oba přístupy jsou použitelné pro řešení této úlohy. Při použití většího datasetu lze určitě dosáhnou i podstatně lepších výsledků. Dále lze modulární program vylepšit, zejména v oblasti detekce a trasování. Nakonec je dobré zmínit i možnost přiblížení obrazu na základě detekce v předchozích snímcích, což může zlepšit klasifikační výsledky. U modulárního programu by případná implementace přibližování obrazu byla složitější, protože je vyžadovaná statická scéna.

Použité přístupy lze provozovat i na obyčejném *PC*. Dedikovaná grafika je výhodou. Při použití kompaktního hardwaru lze doporučit jednodeskový počítač *Jetson Nano Developer Kit*. Jako kamera může posloužit i lepší webkamera. Pokud bude proveden výběr kamery za účelem zlepšení výsledků, je situace složitější a nelze vybrat univerzální výrobek, záleží na konkrétních požadavcích a umístění přípravku. Pro většinu případů je dostatečná kamera *IMX219-77 8Mpx*.

Literatura

[1] SIMONYAN, Karen a Andrew ZISSERMAN. Very Deep Convolutional Networks for Large-Scale Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[2] LOUKADAKIS, Manolis, José CANO a Michael O'BOYLE. Accelerating Deep Neural Networks on Low Power Heterogeneous Architectures. MULTIPROG, 2018.

[3] SZEGEDY, Christian, Wei LIU, Yangqing JIA, Pierre SERMANET, Scott REED, Dragomir ANGUELOV, Dumitru ERHAN a Vincent VANHOUCKE. Going Deeper with Convolutions. SIMONYAN, Karen a Andrew ZISSERMAN. Very Deep Convolutional Networks for Large-Scale Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

[4] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Deep Residual Learning for Image Recognition. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)., 2016.

[5] GIRSHICK, Ross, Jeff DONAHUE, Trevor DARRELL a Jitendra MALIK. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[6] UIJLINGS, J.R.R., K.E.A. VAN DE SANDE, T. GEVERS a A.W.M. SMEULDERS. *Selective Search for Object Recognition*. 2012. International Journal of Computer Vision.

[7] *Caffe* [online]. Yangqing Jia [cit. 2021-01-03]. Dostupné z: https://caffe.berkeleyvision.org

[8] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012. Neural Information Processing Systems.

[9] GIRSHICK, Ross. *Fast R-CNN*. 2015. Proceedings of the IEEE International Conference on Computer Vision (ICCV).

[10] REN, Shaoqing, Kaiming HE, Ross GIRSHICK a Jian SUN. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[11] HE, Kaiming, Georgia GKIOXARI, Piotr DOLLAR a Ross GIRSHICK. *Mask R-CNN*. 2017. Proceedings of the IEEE International Conference on Computer Vision (ICCV).

[12] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK a Ali FARHADI. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[13] SZEGEDY, Christian, Wei LIU, Yangqing JIA, et al. Going Deeper With Convolutions. 2015. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[14] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU a Alexander C. BERG. *SSD: Single Shot MultiBox Detector* [online]. Springer, Cham, 2016 [cit. 2021-01-03]. Dostupné z: https://doi.org/10.1007/978-3-319-46448-0_2

[15] 2020 ROBOFLOW, INC. *Computer Vision Annotation Formats* [online]. In: . [cit. 2021-01-03]. Dostupné z: https://roboflow.com/formats

[16] SVANSTRÖM, Fredrik. *Drone Detection and Classification using Machine Learning and Sensor Fusion*. 2020. Dissertation. Halmstad University.
Dataset [cit. 2021-05-11]. Dostupné z: https://github.com/DroneDetectionThesis/Drone-detection-dataset

[17] Make Sense [online]. Piotr Skalski [cit. 2021-05-11]. Dostupné z: https://www.makesense.ai

[18] HANSEN, Oskar Liset Pryds, Jens-Christian SVENNING, Kent OLSEN, Steen DUPONT, Beulhah H. GARNER, Alexandros IOSIFIDIS, Benjamin W. PRICE a Toke T. HØYE. Image data used for publication "Species-level image classification with convolutional neural network enable insect identification from habitus images [online]. 2019 [cit. 2021-5-11]. Dostupné z: https://doi.org/10.5281/zenodo.3549369

[19] PikiWiki Israel 43324 Wildlife and Plants of Israel.jpg: Attribution 2.5 Generic (CC BY 2.5) [online]. [cit. 2021-5-11]. Dostupné z: https://commons.wikimedia.org/wiki/User:Geagea/Israel/2015_June_14#

[20] Cross winds Luton airport. Youtube [online]. 2012 [cit. 2021-5-11]. Dostupné z: https://www.youtube.com/watch?app=desktop&v=b4f1HYZMpSk&ab_channel=TonyRimmington

[21] TYPHOON H FLY BUKIT MARAS BATU RAKIT. Youtube [online]. 2018 [cit. 2021-5-11]. Dostupné z: https://www.youtube.com/watch?v=Ear7d2vvaq0&ab_channel=PokMan

[22] Буксировка вертолёта. Youtube [online]. 2013 [cit. 2021-5-11]. Dostupné z: https://www.youtube.com/watch?v=DskevtTIFWg&ab_channel=%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B5%D0%B9%D0%93%D0%BE%D0%BB%D1%83%D0%B1

[23] Голубое небо весны. FotoPrizer [online]. 2017 [cit. 2021-5-11]. Dostupné z: https://www.fotoprizer.ru/foto-gallery/one-foto/117987/?q=462&nf=117987

[24] КРАШ-ТЕСТ. WLtoys V383 ... 3D квадрокоптер с коллективным шагом. Youtube [online]. 2015 [cit. 2021-5-11]. Dostupné z: https://www.youtube.com/watch?v=VerYbuKJOZ8&ab_channel=RCBuyer%2FRC%D0%BE%D0%B1%D0%B7%D0%BE%D1%80%D1%8B

[25] *OpenCV* [online]. OpenCV team [cit. 2021-01-03]. Dostupné z: https://opencv.org

[26] *PyTorch* [online]. Facebook [cit. 2021-01-03]. Dostupné z: https://ai.facebook.com/tools/pytorch

[27] *TensorFlow:* Google Brain team [online]. [cit. 2021-01-03]. Dostupné z: https://www.tensorflow.org

[28] JUREČKA, Tomáš. Detekce a klasifikace létajících objektů [online]. Brno, 2021 [cit. 2021-5-11]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/130734 Semestrální práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Ilona Janáková.

[29] JANOUSEK, J., J. NOVOTNY, P. MARCON, A. SIRUCKOVA a R. KADLEC. Algorithms for Flying Object Detection. PIERS — Toyama, 2018. Faculty of Electrical Engineering and Communication, Brno University of Technology

[30] CANU, Sergio. Object Tracking with OpenCV and Python [online]. [cit. 2021-5-11]. Dostupné z: https://pysource.com/2021/01/28/object-tracking-with-opencv-and-python

[31] CHILAMKURTHY, Sasank. TRANSFER LEARNING FOR COMPUTER VISION TUTORIAL [online]. [cit. 2021-5-11]. Dostupné z: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

[32] BABENKO, Boris, Ming-Hsuan YANG a Serge BELONGIE. Robust Object Tracking with Online Multiple Instance Learning. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2011.

[33] JOCHER, Glenn. *Train Custom Data Tutorial* [online]. 2020 [cit. 2021-01-03]. Dostupné z: https://github.com/ultralytics/yolov5/issues/12

[34] Deep learning with Raspberry Pi and alternatives in 2021 [online]. [cit. 2021-5-11]. Dostupné z: https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html

[35] IMX219-77 Camera. Waveshare [online]. [cit. 2021-5-11]. Dostupné z: https://www.waveshare.com/wiki/IMX219-77_Camera

[36] IMX477 12.3MP Camera. Waveshare [online]. [cit. 2021-5-11]. Dostupné z: https://www.waveshare.com/wiki/IMX477_12.3MP_Camera

[37] Arducam 12MP IMX477 Motorized Focus High Quality. UCTRONICS [online]. [cit. 2021-5-11]. Dostupné z: https://www.uctronics.com/imx477-12mp-high-quality-camera-motorized-focus-autofocus.html

[38] ROZANTSEV, A., V. LEPETIT a P. FUA. *Detecting Flying Objects Using a Single Moving Camera*. 2017. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[39] *YOLOv5* [online]. In: . [cit. 2021-01-04]. Dostupné z: https://github.com/ultralytics/yolov5

SEZNAM ZKRATEK

AI	Umělá inteligence
CNN	Konvoluční neuronová síť
FN	False negative
FP	False positive
FPS	Počet snímků za sekundu
GPU	Grafický procesor Aktivační funkce neuronu
MOG	Gaussova směs Aktivační funkce neuronu
RGB	Barevný model (červená, zelená, modrá)
RPN	Konvoluční neuronová síť pro detekci a označení objektů
SGD	Stochastický gradient (iterační metoda pro optimalizaci)
SSD	Single Shot MultiBox Detector (model pro detekci a klasifikaci v reálném čase)
SVM	Metoda podpůrných vektorů
TN	True negative
TP	True positive
VGG	Visual Geometry Group (vychází z názvu oddělení) (Konvoluční neuronová síť)
YOLO	You only look once (model pro detekci a klasifikace v reálném čase)
ZF	Zeiler and Fergus (jména autorů) (Konvoluční neuronová síť)

SEZNAM PŘÍLOH

složka	podsložka	popis
YOLO	trénování	výstupy z trénování
	testování	ukázka výsledků testování
Modulární_program	ukázka	video-ukázka běhu programu
Použité_skripty	dataset	skripty pro tvorbu datasetu
	úprava	skripty pro úpravu obrázků
	hodnocení	skripty pro hodnocení modelů a programů