FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Zařízení pro rozpoznávání obrazu v reálném čase

Studijní program: B0613A140005 – Informační technologie
Studijní obor: B0613A140005AI – Aplikovaná informatika
Autor práce: **Hynek Václav Svobodný**
Vedoucí práce: Ing. Jana Kolaja Ehlerová, Ph.D.

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Zařízení pro rozpoznávání obrazu v reálném čase

Jméno a příjmení:	**Hynek Václav Svobodný**
Osobní číslo:	M21000138
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav nových technologií a aplikované informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Proveďte rešerši systémů pro rozpoznávání obrazu s přihlédnutím pro rozpoznávání ptáků.
2. Navrhněte a implementujte automatický systém snímkování ptačího krmítka a zpracování obrazu pomocí rozpoznávání, otestujte vhodné metody.
3. Vytvořte systém pro zobrazení výsledků v reálném čase s možností notifikací a statistického vyhodnocení pozorování.
4. Testujte systém pro nestandardní situace a proveďte jejich ošetření.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 – 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image processing, analysis, and machine vision*. Fourth Edition. Australia: Cengage Learning, [2015]. ISBN 978-1-133-59369-0.

[2] SHRESTHA, Ajay a Ausif MAHMOOD. Review of Deep Learning Algorithms and Architectures. IEEE Xplore [online]. 22. duben 2019 [vid. 28. květen 2023]. Získáno z: https://ieeexplore.ieee.org/abstract/document/8694781/

[3] Cornell University. Merlin Bird Id – home. Merlin Bird ID – Free, instant bird identification help and guide for thousands of birds [online]. 2023 [vid. 28. květen 2023]. Získáno z: https://merlin.allaboutbirds.org/

Vedoucí práce:	Ing. Jana Kolaja Ehlerová, Ph.D. Ústav nových technologií a aplikované informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 19. října 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.
Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.
Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.
Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.
Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.
Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.
Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

14. 5. 2024 Hynek Václav Svobodný

Zařízení pro rozpoznávání obrazu v reálném čase

Abstrakt

Tato práce pojednává o hardwarově-softwarovém řešení, které v reálném čase snímkuje ptačí krmítko, používá strojové učení k rozpoznání ptáků v obraze a určení jejich druhu a vede záznam. Dále je v práci popsána webová aplikace, která zpřístupňuje statistiky a živý stream s anotacemi nalezených ptáků. V práci je dále popsán vlastní model pro klasifikaci druhů ptáků vyskytujících se v dané oblasti.

Klíčová slova: hluboké neuronové sítě, ptáci, Raspberry Pi, rozpoznávání obrazu

Real-time image recognition device

Abstract

This paper discusses a hardware-software solution that takes real-time images of a bird feeder, uses machine learning to recognize birds in the image and determine their species and keeps a record. The paper also describes a web application that makes available statistics and a live stream annotating the birds found. The thesis also describes a custom model for classifying bird species occurring in a given area.

Keywords: deep neural networks, birds, Raspberry Pi, image recognition

Poděkování

Chtěl bych poděkovat své babičce Jiřině Bajgarové, která trpělivě poskytovala krmení ptákům do krmítka, díky čemuž jej velmi hojně navštěvovali a bylo je tak možné sledovat prakticky kdykoliv během dne.
Dále bych chtěl poděkovat všem členům rodiny za psychickou podporu při psaní této práce.

Obsah

Seznam zkratek

DNN	deep neural network
CNN	convolutional neural network
R-CNN	region-based convolutional neural network
YOLO	You Only Look Once
YOLOv8	You Only Look Once version 8
YOLOv3	You Only Look Once version 3
API	Application Programming Interface
RTSP	Real-Time Streaming Protocol
HLS	HTTP Live Streaming
WebRTC	Web Real-Time Communication
COCO	Common Objects in Context
CSI	Camera Serial Interface
HDMI	High-Definition Multimedia Interface
HAT	hardware attached on top
IP67	Ingress Protection 67
VPN	virtual private network
IP	Internet Protocol
REST API	Representational State Transfer Application Programming Interface
NAT	Network Address Translation
SSH	Secure Shell
UDP	User Datagram Protocol
CLI	Command-line interface
LTS	Long-term support
NMS	Non-maximum suppression
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
WSGI	Web Server Gateway Interface

1 Úvod

Již před započetím tohoto projektu mě zaujala problematika rozpoznávání ptáků v reálném čase a identifikace jejich druhů. Zaujal mě projekt dobrovolného sčítání ptáků, které lidé každoročně provádějí ručně a nazývá se Ptačí hodinka, což pomáhá ornitologům získat informace o stavu různých druhů na různých lokalitách. Myslel jsem si, že tuto činnost by bylo možné automatizovat pomocí zařízení schopného rozpoznávat ptáky na záběrech míst, kde se vyskytují, a následně určovat jejich druhy pomocí strojového učení.

Dalším motivujícím faktorem pro výběr tohoto tématu byla situace na naší rodinné chalupě v Jizerských horách, kde se nachází prostorné krmítko, jež je často navštěvované místními ptáky. Je možné je tam snadno pozorovat a vytvořit o nich statistiky.

V této práci je představeno hardwarově-softwarové řešení, které v reálném čase rozpoznává ptáky a určuje jejich druh pomocí hlubokých neuronových sítí a vede o nich statistiky. Tyto statistiky společně s živým video streamem jsou zpřístupněny na webové aplikaci.

2 Rešerše

2.1 Ptačí hodinka

Ptačí hodinka je projekt zaštítěný Českou společností ornitologickou a představuje každoroční sčítání ptáků na krmítkách. Do sčítání se může zapojit kdokoliv, i bez předchozích zkušeností, projekt je určen všem zájemcům o ptačí dění v okolí. Projekt existuje od oku 2019.

Ptačí hodinka si dává za cíl dlouhodobě sledovat zimování ptáků v České republice a zjišťovat, co ovlivňuje jejich výskyt a chování. Dalším cílem je zapojení veřejnosti do vědeckého výzkumu, což u lidí napomáhá k budování vztahu k ptákům a přírodě.

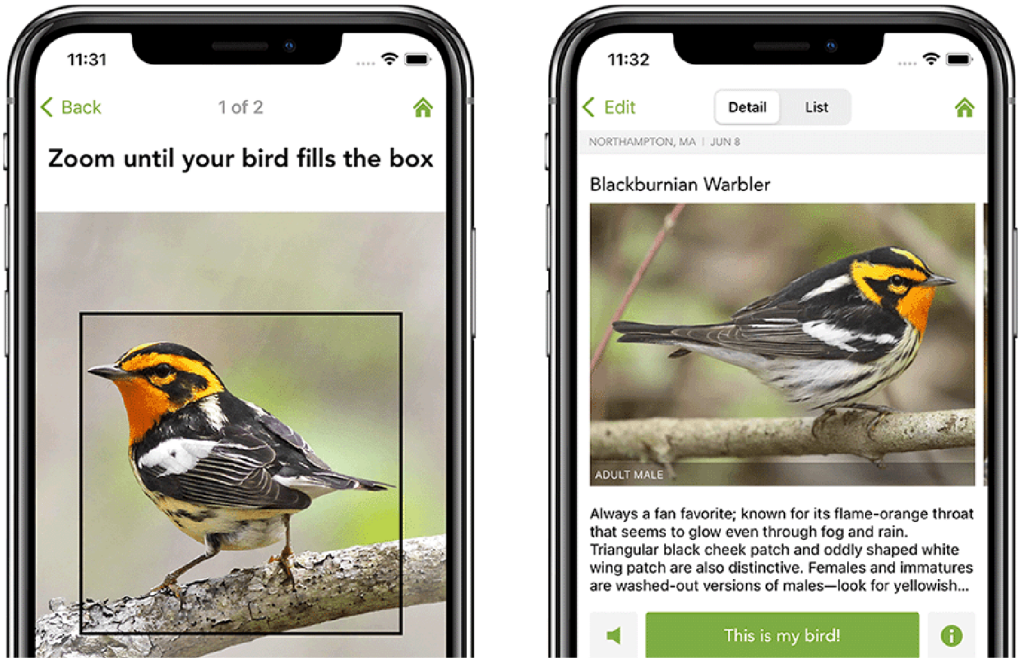
Kromě české Ptačí hodinky existují podobná sčítání i v zahraničí. Česká společnost ornitologická úzce spolupracuje s ornitologickými společnostmi z Bavorska, Rakouska, Švýcarska, od roku 2024 nově ze Slovenska a Srbska. Díky této spolupráci je možné výsledky porovnávat přeshraničně. [4]

2.2 Úvod do rozpoznávání obrazu strojovým učením

Hluboké neuronové sítě (DNN) simulují strukturu lidského mozku tím, že obsahují uzly jako neurony, propojené do vrstev. Každý uzel má vstupy a výstupy, přičemž spojení mezi nimi má přiřazenou váhu, a výstup je ovlivněn aktivační funkcí. Trénovací algoritmy hledají optimální hodnoty vah pomocí trénovacích dat, což umožňuje DNN naučit se rozpoznávat složitější vlastnosti dat. [17]

Pro rozpoznávání obrazu se používají Konvoluční neuronové sítě (CNN). Obsahují vrstvy konvoluce, což umožňuje postupně extrahovat stále složitější vlastnosti z obrazových dat. Na konci mají plně propojenou vrstvu a vrstvu normalizace. [17]

Pro detekci objektů v obrazech se často používají algoritmy a modely postavené na CNN, jako jsou například Region based CNN (R-CNN), Fast Region based CNN a Faster Region based CNN. [5] V této práci je použitý algoritmus YOLO ve verzi 8 (YOLOv8), přičemž v počátečních fázích vývoje se používal YOLO ve verzi 3 (YOLOv3).



Obrázek 2.1: Ukázka práce s Merlin Bird ID

2.3 Projekty rozpoznávání ptáků

Existuje řada projektů, které využívají rozpoznávání ptáků nejen za pomoci obrázků. Jednotlivé projekty se od sebe liší mnoha aspekty, mezi které patří např. platforma, nebo výskyt dalších funkcí. Důležitým aspektem v porovnání k mé práci je rozpoznávání živého streamu a ne pouhé fotografie, dále pak vedení statistik.

Prvním takovým projektem je mobilní aplikace Merlin Bird ID. Je dostupná pro Android a iOS. Využívá databázi eBird, která shromažďuje data od přispěvatelů a dokumentuje výskyt druhů ptáků v různých oblastech. Aplikace umožňuje určovat ptáky pomocí informací o jejich vzhledu, chování, polohy výskytu, ale také pomocí zvuku a fotografie. [3] Právě rozpoznání podle fotografie je nejbližší této práci, ale stále je to rozpoznávání fotografií na pokyn uživatele a ne samostatné sledování ptačího dění v určitém místě. Ukázku práce s rozpoznáváním podle fotografie je možné vidět v obrázku 2.1.

Dalším takovým projektem je Picture Bird, jenž představuje aplikaci dostupnou na platformách iOS, Android, Mac a Windows. Tato aplikace umožňuje rozpoznávání ptáka podle obrázku nebo zvuku. Kromě toho v rámci projektu existuje encyklopedie ptáků a je také umožněna expertní konzultace s ornitology. [14] Zde je rozpoznání podle obrázku opět nejbližší této práci, ale opět se nejedná o soustavné sledování určitého místa.

2.4 Použité technologie

2.4.1 Technologie použité pro strojové učení

Pro trénování vlastního modelu pro klasifikaci obrázků jsem použil TensorFlow a v něm vestavěný Keras, což je vysokoúrovňové API, které se dá použít i s jinými frameworky. [12]

Pro detekci objektů v obrázku používám technologii YOLO, ve finální podobě práce verzi 8, kde používám předtrénovaný model Ultralytics YOLOv8 [13], jehož balík poskytuje i možnost exportu do jiných formátů [2].

Pro běh všech modelů jsem použil framework TensorFlow Lite, který je optimalizován pro běh na mobilních a vestavěných zařízeních, popdporuje hardwarovou akceleraci, je multiplatformní a podporuje používání z několika programovacích jazyků, včetně Pythonu použitého v této práci. [11]

2.4.2 Technologie použité pro streamy

Pro přeposílání video streamů jsem použil program MediaMTX, který umožňuje odesílat a konzumovat streamy pomocí protokolů např. RTSP, HLS, WebRTC, je možné jej nazývat "mediálním routerem". V případě, že je MediaMTX nastaven jako proxy vedoucí na jiný stream, existuje zde možnost připojení se k cílovému streamu pouze, pokud má proxy alespoň jednoho klienta připojeného, jinými slovy se data nepřenášejí, pokud je nikdo nekonzumuje. Program je napsán v jazyce Go. [1]

2.4.3 Technologie použité pro webovou aplikaci

Backend webové aplikace je napsaný v jazyce Python a využívá framework Django, který pokrývá časté aspekty vývoje webových aplikací [7], jako je databáze [6], přihlašování [9] nebo lokalizace [8].

Jako webový server jsem z důvodů uvedených v kapitole 5.2.1 zvolil mod_wsgi-express, což je separátní instance Apache, na které běží mod-wsgi, tedy propojení s Python kódem.

2.5 Výchozí zdroje pro tuto práci

Během vývoje softwaru této práce byl vytvořen vlastní kód, avšak pro detekci objektů v obrazech, klasifikaci obrázků a trénování modelu klasifikace obrázků jsem původně čerpal z dostupných online zdrojů.

Pro detekci objektů v obrazech jsem využil článek [15], který ukazuje detekci objektů ve videu pomocí modelu YOLOv3 natrénovaného na datasetu COCO. Jelikož byl kód začleněn do objektově-orientované struktury, existují už pouze jeho fragmenty na určitých místech zpracování dat nebo práce s modely.

Pro klasifikaci obrázků jsem se inspiroval článkem [16], ve kterém je popsána tvorba webové aplikace pro rozpoznání zdravého/nezdravého listu rostliny. Použil

jsem kód pro klasifikaci obrázků z tohoto článku, který jsem integroval do této práce, přičemž tedy podobně jako v případě zmíněném výše splynul s ostatním kódem.

Při trénování modelu pro klasifikaci obrázků jsem využil článek popisující metodu transfer learning na modelu s architekturou EfficientNet. Tento článek podrobně vysvětluje postup trénování modelu, který je v základu již natrénován na datasetu ImageNet, na vlastních datech. [10] Kód trénování nebyl začleněn do objektového návrhu, a tak se v pozměněné podobě vyskytuje a používá i v konečné podobě této práce.

Část práce byla odvedena v mém bakalářském projektu. [18]

3 Hardware

Pro tuto práci jsem použil hardware, který by se dal rozdělit na dvě části. První skupina je počítač Raspberry Pi s kamerou, které jsou umístěny u krmítka, dále existuje virtuální server, jehož hardware však nevlastním a neovládám, protože je poskytován jako služba na hostingu.

3.1 Krmítko

K ptačímu krmítku jsem umístil hardware, který zajišťuje jeho snímkování a následné zpracování obrazu. Sestává z počítače Raspberry Pi 4 a kamerového modulu Raspberry Pi Camera 3.

Počítač je s kamerou propojen přes rozhraní CSI, jehož kabel je příliš křehký na to, aby se s ním procházelo zdí a vedl se vně budovy. Proto je CSI spojení přemostěno přes HDMI kabel, k čemuž jsem využil pár převodníků mezi CSI a HDMI. Tyto jsou umístěny co nejblíže koncovému zařízení, takže ven z ochranné krabičky jak počítače, tak kamery, vede již jen HDMI kabel.

3.1.1 Počítač Raspbery Pi 4

Počítač Raspberry Pi 4 použitý v této práci slouží k získávání a rozpoznávání obrazových dat z kamery. Informace o rozpoznaných objektech eviduje a odesílá na server, poskytuje také stream, ve kterém jsou anotované nalezené objekty.

Fotografie umístění počítače Raspberry Pi je možné vidět v obrázcích 3.1, 3.2 a 3.3

Počítač je umístěn ve vnitřních prostorech nedaleko krmítka v ochranné krabičce. V krabičce je umístěn také převodník CSI-HDMI, kvůli čemuž jsem vybral krabičku, která podporuje umísťování HAT modulů, je tak vyšší a obsahuje velký otvor pro vstupy a výstupy HAT Modulů. Převodník jsem suchými zipy připevnil na strop krabičky a tímto velkým otvorem vychází již HDMI kabel. Nevýhodou tohoto řešení je nutnost převodník přidržovat při manipulaci a fakt, že CSI kabel je zkroucen v blízkosti procesoru počítače. Počítač Raspberry Pi je napájen pomocí adaptéru ze sítě.

Raspberry Pi je připojeno k Internetu přes lokální síť, ke které je připojeno přes Wi-Fi. V daném místě není signál Wi-Fi příliš silný, ale obrazová data, která jsou nejobjemnější, lze stále přenášet v dostatečné kvalitě pro zobrazení uživateli.

Obrázek 3.1: Umístění Raspberry Pi

Obrázek 3.2: Detail Raspberry Pi – uzavřeno

Obrázek 3.3: Detail Raspberry Pi – otevřeno

3.1.2 Kamerový modul

Pro samotné získávání obrázků je u krmítka umístěna kamera Raspberry Pi Camera v3.

Kamera je umístěna na kovové konstrukci v ochranné krabičce, která splňuje specifikaci IP67. Uvnitř krabičky se nachází také převodník HDMI-CSI, se kterým je kamera spojena šrouby díky jejich společnému půdorysu. Tvoří tak celek, ke kterému je připojen HDMI kabel, jenž vede z krabičky ven. Onen celek je přes suchý zip připevněn ke korkové zátce, která slouží jako podložka, aby byla kamera vhodně umístěna v rámci krabičky. Celek je odnímatelný.

Během práce jsem se potýkal s problémy právě s kamerou, zejména její ochranou. Na začátku práce zde byla umístěna kamera Arducam 16Mpx v jiné krabičce. Několik kamer bylo zničeno vlhkostí, kterou jsem se snažil zmírnit pomocí sáčků na pohlcování vlhkosti. Po výměně krabičky jse zjistil, že byl v horní průchodce vytvořen otvor, pravděpodobně vinou některého z ptáků. Po výměně již poslední kamera nevykazuje žádné chyby a díky průhlednému víku krabičky je vidět, že je uvnitř suchá. Fotografie kamery lze nalézt v obrázcích 3.4 a 3.5.

3.1.3 Akcelerátor Google Coral

Pro lepší a efektivnější fungování rozpoznávací aplikace jsem plánoval použít Google Coral, který obsahuje koprocesor EdgeTPU pro akceleraci neuronových sítí a existuje pro něj podpora v frameworku TensorFlow Lite. Problém se vyskytoval u detekčního modelu, který se nedařilo načíst do akcelerátoru. Proto akcelerátor není použitý ve finální podobě práce.

3.2 Virtuální server na hostingu

V této práci se kromě lokálního hardware u krmítka používá také virtuální server provozovaný na hostingu. Na serveru běží webová aplikace zpřístupňující statistiky a živý stream. Kromě toho zde běží VPN server, který zajišťuje konektivitu mezi serverem, počítačem u krmítka a případně dalšími zařízeními, např. mými počítači, které potřebují zabezpečený a přímý přístup k řadě služeb na serveru a počítači u krmítka.

Fakt, že server se nachází na hostingu a je virtuální, znamená, že jeho hardware nemohu přímo ovládat. Má však oproti serveru umístěnému doma či přímo na chalupě u krmítka několik výhod:

- Zajištění neustálého a stabilního přívodu elektrické energie.
- Zabezpečení hardware proti případnému požáru nebo jiným haváriím vzniklým v nepřítomnosti osob v bytě nebo budově.
- Zajištění stabilního připojení k internetu s veřejnou IP adresou. Veřejná IP adresa není standardní pro domácí sítě.

Obrázek 3.4: Kamera – detail

Obrázek 3.5: Kamera s krmítkem

4 Infrastruktura, operační systém a konfigurace

Pro tuto práci a další moje projekty se využívá infrastruktura, jejímž středem je virtuální server umístěný na hostingu a VPN síť, která umožňuje a zabezpečuje komunikaci s koncovými zařízeními.

Schéma infrastruktury je možné nalézt v diagramu 4.1.

4.1 VPN

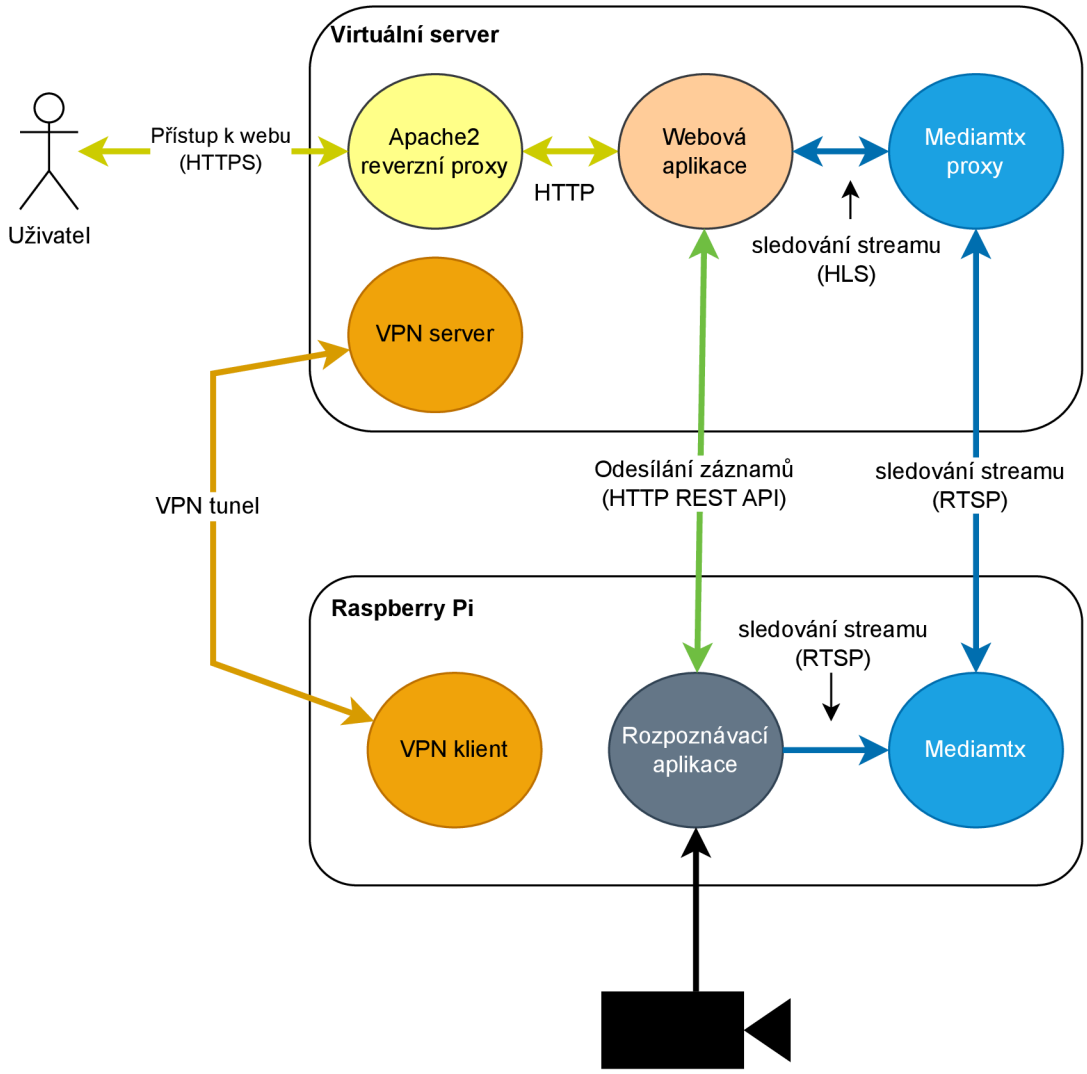
Počítač Raspberry Pi a server jsou propojeny prostřednictvím sítě VPN. Přímé připojení zvenku do lokální sítě není možné, protože domácí sítě často využívají Network Address Translation (NAT), což znamená, že veřejná IP adresa není běžně dostupná a musí být obvykle získána jako doplňková služba. Díky VPN je však možné přistupovat k počítači i mimo místní síť. Osobní zařízení se mohou také připojit k VPN a získat tak přístup k počítači Raspberry Pi, i když se nenachází ve stejné lokální síti.

Další důvod, proč používám ve své infrastruktuře VPN, i když komunikuji pouze se serverem, který je veřejně dostupný, je bezpečnost. VPN umožňuje bezpečný přístup k serveru i k počítači u kamery, a tím pádem není nutné řešit šifrování nebo riziko prolomení hesla u služeb, které se používají pouze uvnitř VPN. Většina portů, které jsou určeny pouze pro interní použití a správu, je otevřená pouze do VPN.

Klienti VPN jsou nejen počítač Raspberry Pi, ale i moje osobní zařízení, a další zařízení související s jinými projekty. Připojují se k VPN přes otevřený port na serveru, přes který navážou šifrovaný tunel. Tento tunel vytváří zdánlivé přímé spojení mezi klientem a serverem, což umožňuje oboustrannou komunikaci a server funguje též jako zprostředkující síťový prvek pro komunikaci mezi jednotlivými klienty.

4.2 Virtuální server

Operačním systémem serveru je Ubuntu Server 20.04 LTS, který jsem vybral, protože jej znám a je to jedna ze standardních linuxových distribucí používaných na servery. Jeho primárním rozhraním je příkazový řádek, ke které se připojuji pomocí SSH. Grafické uživatelské rozhraní jsem doinstaloval pro projekty vyžadující desktopové aplikace a pro plný přístup ke službám uvnitř VPN z počítačů, které nemají přístup do VPN.



Obrázek 4.1: Schéma infrastruktury

OpenVPN server zprostředkovává výše uvedenou síť VPN, připojení je k dispozici na UDP portu 1194. Instance s danou konfigurací je vedena jako systémová služba.

Na serveru běží webový server Apache, který zpřístupňuje několik webů různých projektů, přičemž jednotlivé weby jsou odlišeny pomocí virtualhostů. Jeden z těchto webů je z této práce a zobrazuje statistiky a živý stream. Pro tento web slouží tato instance webového serveru pouze jako reverzní proxy pro jinou instanci běžící v rámci webové aplikace, viz .

4.3 Raspberry Pi u krmítka

Operačním systémem Raspberry Pi je Raspberry Pi OS, konkrétně jeho 64bitová verze. Volba tohoto operačního systému byla přirozená, protože se jedná o linuxovou distribuci vydávanou výrobcem Raspberry Pi. Jediným jeho uživatelským rozhraním je příkazový řádek, ke kterému se připojuji přes SSH.

Raspberry Pi je připojeno k síti VPN, díky které je dostupné i mimo lokální síť na chalupě, takže je možná interní komunikace se serverem, a také je Raspberry Pi dostupné z mých počítačů, nezávisle na jejich umístění.

Na Raspberry Pi běží aplikace pro rozpoznávání obrazu z kamery. Tato obsahuje instanci mediálního serveru MediaMTX, kde je dostupný výsledný stream, který přes VPN získává webová aplikace na serveru. Z VPN a lokální sítě je však možné se na stream připojit i přímo, např. z přehrávače VLC.

Pro sběr dat ještě na Raspberry Pi dříve běžela aplikace MotionEye, která měla problém se získáváním dat z kamery Arducam 16Mpx, kvůli čemuž jsem aplikaci zvnějšku rozšířil, aby data z kamery byla získávána přes utilitu libcamera-vid, jejíž běh v této situaci nebyl příliš stabilní a docházelo tak k výpadkům. Tyto byly společně s netěsnící krabičkou výraznými překážkami při zprovozňování kamery.

5 Implementace software

Pro tuto práci jsem napsal dvě aplikace. První aplikace rozpoznává obraz pomocí neuronových sítí, druhá je webová aplikace zobrazující stream a zaznamenaná data.

Aplikace pro rozpoznávání obrazu běží na Raspberry Pi a získává obrazová data z kamery a provádí rozpoznávání. Obraz s anotacemi poskytuje jako výstupní stream a zaznamenává statistku rozpoznaných ptáků.

Webová aplikace běží na virtuálním serveru a zobrazuje statistiku výskytu jednotlivých druhů ptáků ve zvoleném čase, zároveň uživatelům s patřičným oprávněním zpřístupňuje živý stream z aplikace pro rozpoznávání obrazu.

5.1 Aplikace pro rozpoznávání obrazu

Pro rozpoznávání ptáků v obrazových datech a ukládání výsledků jsem vyvinul aplikaci v jazyce Python 3, která, jak již bylo uvedeno, běží na počítači Raspberry Pi 4 u krmítka. Je napsaná převážně pomocí objektově orientovaného paradigmatu.

Aplikace sleduje obraz z krmítka a pomocí DNN nachází ptáky v obraze a určuje jejich druh. Poskytuje výstupní stream, ve kterém jsou anotovány polohy a názvy druhů nalezených ptáků. Tyto také zapisuje do databáze společně s časem jejich příchodu a odchodu. Tyto výsledky jsou pak nahrávány na server do webové aplikace, pomocí jejího API.

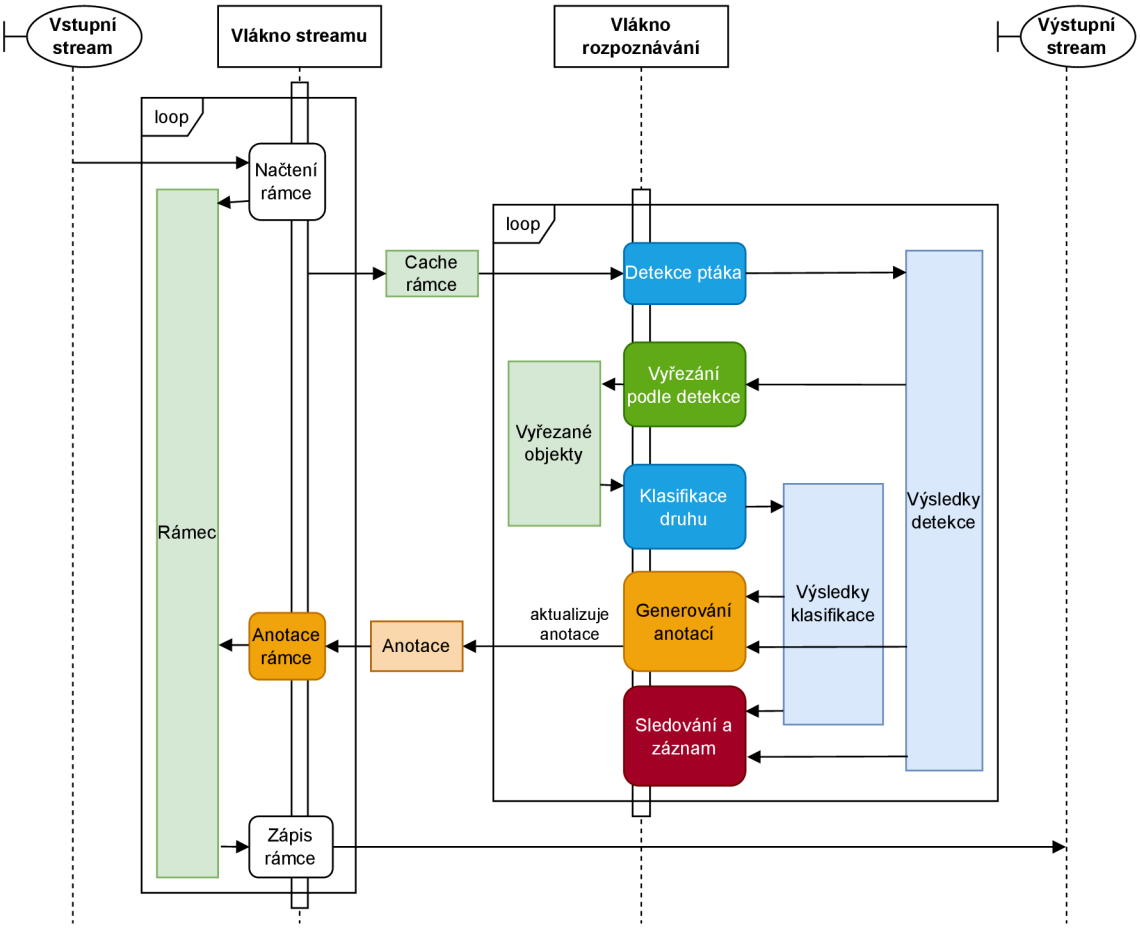
Samotný kód v Pythonu je spouštěn z CLI, přičemž existuje několik vstupních bodů, kde vedle vstupního bodu aplikace, který se používá ve finální podobě práce, vedlejší vstupní body umožňují samostatně spouštět některé menší části z celkové funkcionality, např. pouze provést rozpoznání samostatného obrázku ze souboru.

Celková aplikace kromě samotného programu v Pythonu využívá také program MediaMTX, který soustavně konzumuje stream z aplikace a poskytuje jej pomocí protokolu RTSP do lokální sítě a VPN, kde ji využívá webová aplikace.

V této sekci se vyskytují diagramy tříd, které vždy vystihují určitou vrstvu nebo její část. Pro zjednodušení zde nejsou zobrazeny tovární objekty a CLI, pokud se jich diagram přímo netýká.

5.1.1 Činnosti aplikace v jednotlivých vláknech

Běh aplikace je rozdělen do dvou vláken, kde v prvním vlákně běží rychlé zpracování obrazového streamu za účelem zobrazení v reálném čase, přičemž v druhém vlákně probíhá rozpoznávání ptáků, určování jejich druhů, nastavení anotací pro stream



Obrázek 5.1: Popis činnosti rozpoznávací aplikace

a záznam výsledků. Schéma činností v jednotlivých vláknech a jejich propojení je možné nalézt v diagramu 5.1.

Již zmíněné vlákno pro rychlé zpracování streamu provádí ve smyčce následující sekvenci:

1. získání rámce z vstupního streamu z kamery
2. uložení kopie rámce pro účely rozpoznávání v druhém vlákně
3. anotování rámce poslední dostupnou anotací.

Vlákno pro rozpoznávání a záznam výsledků provádí ve smyčce tyto kroky:

1. zpracování rámce detekčním modelem pro nalezení ptáků v obraze
2. vyřezání obrázků ptáků podle detekce
3. zpracování obrázků ptáků klasifikačním modelem pro určení druhu ptáka
4. anotace nalezených ptáků s určením druhu a pravděpodobností do původního obrázku

5. registrace nalezených objektů ve sledovací vrstvě, viz 5.1.7

6. v případě, že ve sledovací vrstvě existují objekty k zapsání, tyto zapsat

Z popisu vláken je vidět, že rozpoznávání může probíhat s nižší frekvencí než zpracování streamu, při jehož anotaci se využívají vždy výsledky poslední predikce. V praxi jsou tedy v běžícím streamu anotace vždy po určitou krátkou dobu stejné, než jsou změněny.

5.1.2 Obecná charakteristika aplikace

Aplikace se skládá z několika vzájemně propojených vrstev, kde daná vrstva poskytuje určitou funkcionalitu a je možné ji konfigurovat, často i z rozhraní příkazového řádku. Konkrétní funkcionalita je řešena pokud možno obecně, což napomáhá rozšiřitelnosti a výměně implementace některých částí. Závislost jednotlivých vrstev je možné vidět v diagramu 5.2. Je nutné podotknout, že pro zjednodušení v něm jsou obsaženy pouze vazby provozní logiky a ne inicializace nebo využívání obecných rozhraní.

Každá vrstva je tvořena rozhraními, abstraktními a konkrétními třídami a funkcemi, které poskytují danou funkcionalitu. Složitější funkcionalita je rozdělena na menší třídy metodou kompozice. Dané části pak mohou být vytvářeny nezávisle, čímž dochází ke zjednodušení inicializace aplikace.

Při inicializaci jsou v některých případech objekty vytvářeny pomocí továrních objektů nebo funkcí, díky čemuž se oddělí inicializace od vlastní funkcionality, což může být žádoucí, zejména pokud daná funkcionalita obsahuje mnoho konfiguračních parametrů. Pro zjednodušení diagramů tříd nejsou tovární třídy vyobrazeny, pokud se daný diagram nezabývá přímo jejich implementací.

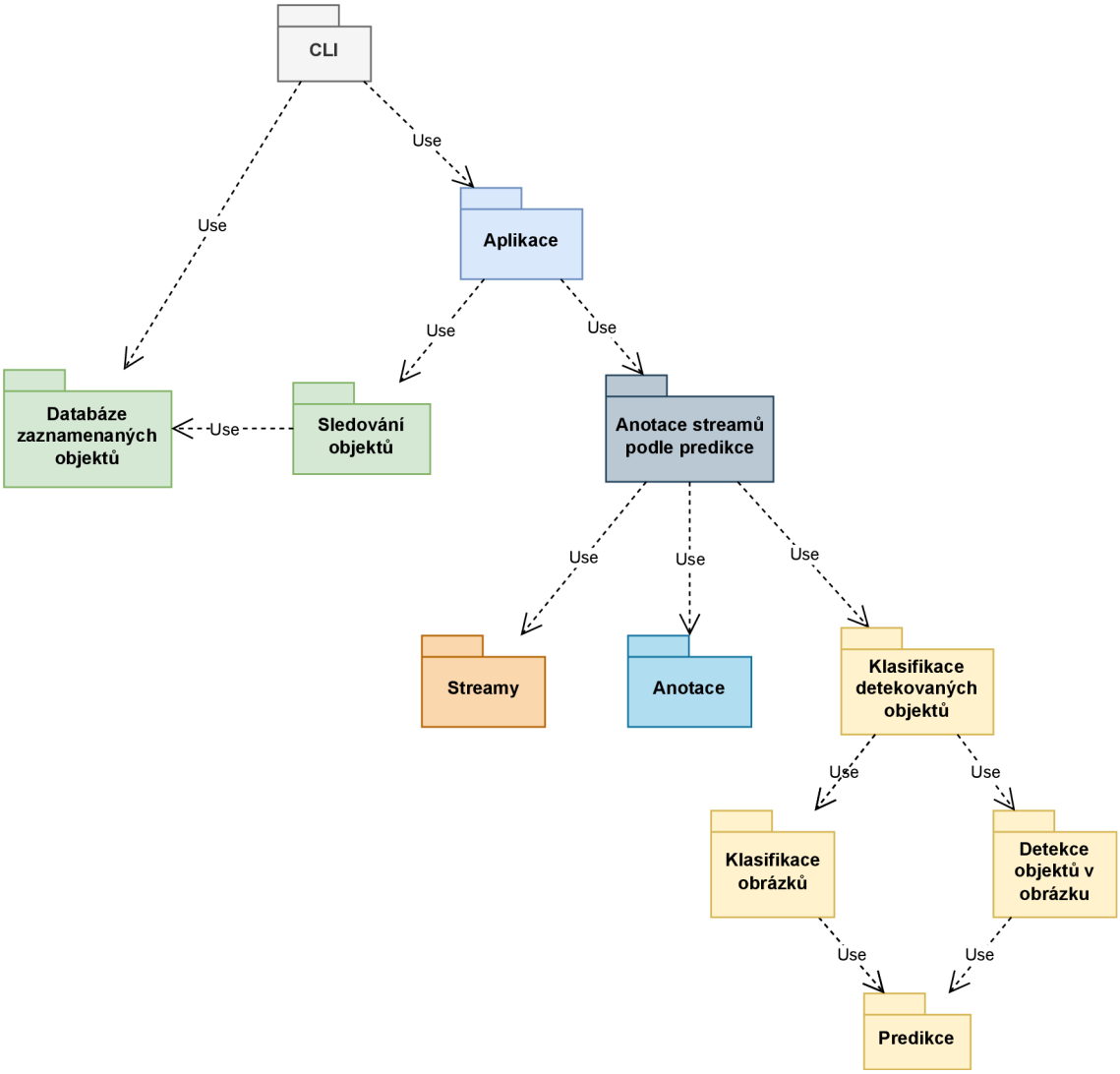
V některých případech je určitá funkcionalita definována obecně a existuje více implementací. Ty jsou definovány nezávisle a jsou automaticky vyhledávány speciálním typem továrních objektů.

5.1.3 Rozpoznávání obrazu

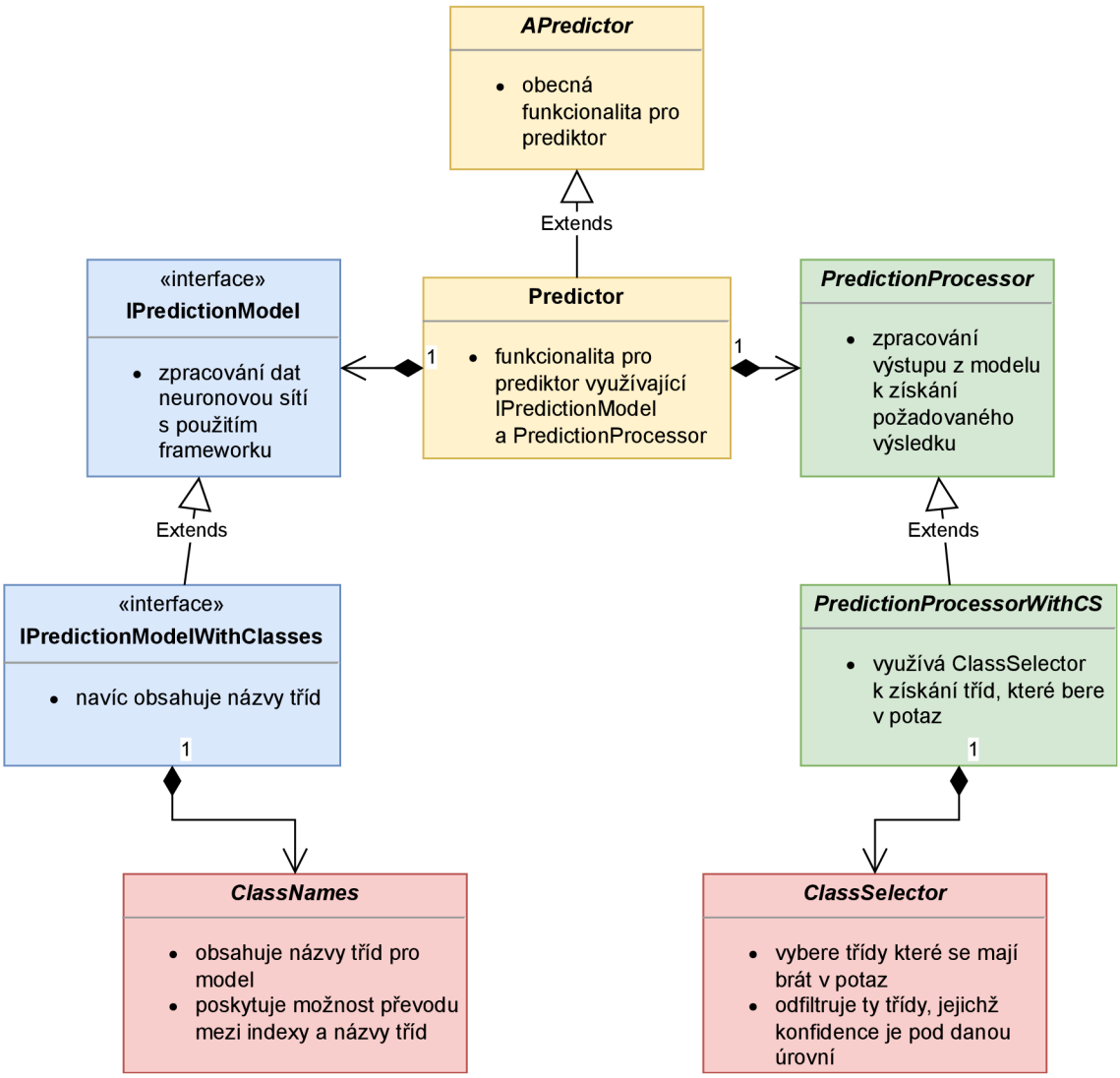
Vyvinul jsem složitou softwarovou vrstvu pro práci s hlubokými neuronovými sítěmi (DNN), která má za úkol abstrahovat tyto sítě z různých frameworků a poskytovat vyšším vrstvám možnost získat relevantní výstupy pro zadané vstupy. Tento přístup jsem zvolil, protože jsem chtěl mít možnost měnit použitý framework dle potřeby ve vývoji.

Pro lepší pochopení této vrstvy je vhodné definovat několik pojmů, které jsou významné nejen pro tuto vrstvu aplikace. Termín ”predikce” slouží jako obecný výraz pro jakoukoli činnost s DNN, která odhaduje realitu. Tato práce se zaměřuje na detekci objektů, klasifikaci obrázků a jejich kombinaci, které lze obecně označit jako různé druhy predikcí.

V diagramu Figure 5.3 je možné vidět obecné vztahy tříd souvisejících s predikcí. Není zde zmínka o konkrétním druhu predikce, jelikož schéma tříd souvisejících s konkrétními druhy je k nalezení v diagramech 5.5, 5.6 a 5.6.



Obrázek 5.2: Diagram vrstev rozpoznávací aplikace



Obrázek 5.3: Obecný diagram tříd vrstvy rozpoznávání obrazu

Na začátku vývoje této práce jsem čerpal inspiraci z několika tutoriálů, pomocí kterých jsem vytvořil jednoduché skripty pro provádění predikcí, tedy klasifikaci a detekci. Tyto počáteční skripty nebyly příliš obecné a nebyly napsány s důrazem na objektově orientované programování. Postupem času jsem však vyvinul mnohem obecnější kód, kde fragmenty původního kódu se v současném zdrojovém kódu nacházejí na různých místech tohoto strukturovaného kódu, především u ovládání neuronových sítí v daném frameworku a zpracování dat těsně před nebo po zpracování modely. Častěji však byl původní kód zcela nahrazen, protože nový kód byl rozdělen do mnoha tříd.

Cílem této vrstvy je poskytnout velmi obecné rozhraní, a proto je velká část kódu napsána tak, aby byla použitelná pro jakýkoliv druh predikce. Konkrétní vstupní i výstupní typy jsou generické a jsou upravovány pro konkrétní druh predikce. Oba konkrétní druhy predikce použité v této práci mají jako vstup obrázek reprezentovaný vícedimenzionálním polem NumPy. Výstupní typy jsou odlišné, což vyplývá z charakteru daného druhu predikce.

Vrstva se dále dělí na modely, které zajišťují ovládání specifického frameworku, dále pak na class selectory, které slouží k obecnému vybírání tříd v predikci bez ohledu na druh predikce, a nakonec prediktory, které využívají výše uvedené společně se zprcováním výstupních dat pro zajištění celkové funkcionality.

Modely

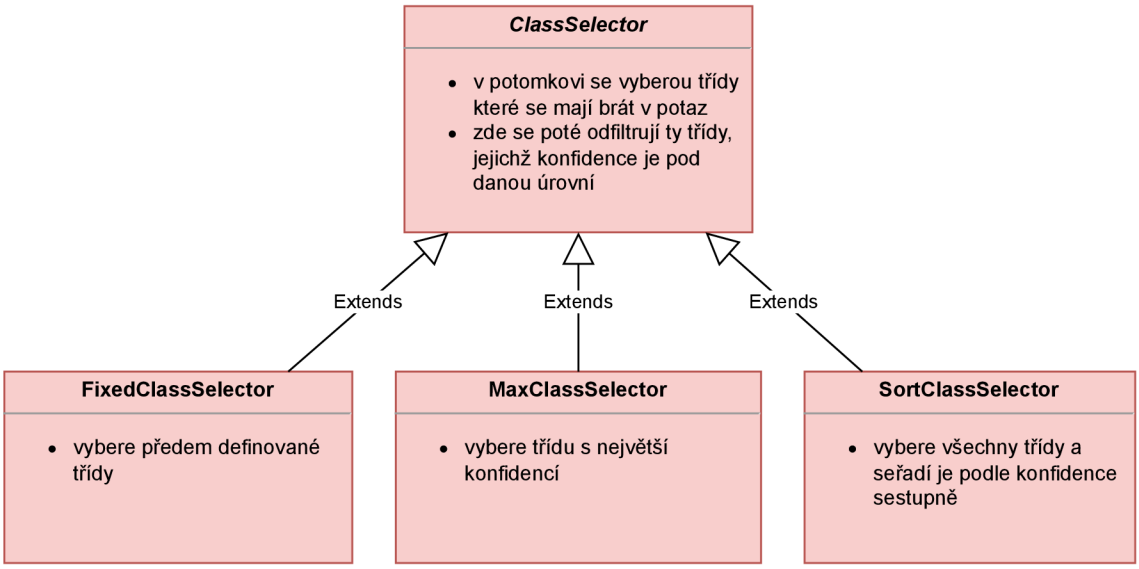
Vrstva pro práci s hlubokými neuronovými sítěmi definuje modely jako třídy, které vytvářejí a zapouzdřují instanci modelu z konkrétního frameworku. Tyto třídy mají jednotné vnější rozhraní pro určitý typ predikce, což umožňuje mít implementaci modelu nezávislou na zpracování výstupu pro získání potřebných dat. Je však nezbytné, aby vstupní a výstupní datový typ byl kompatibilní.

Hierarchie modelů je navržena dvoucestně, kdy cestou implementace rozhraní model odpovídá danému typu predikce (zde klasifikace nebo detekce). Cestou abstraktních tříd mohou modely různých typů predikce ale stejného frameworku sdílet části, které jsou pro všechny modely frameworku společné.

Modely, které pracují s třídami (v této práci všechny), mají sdílenou funkcionalitu načítání názvů tříd, které jsou uloženy a načítány společně s modelem, protože názvy tříd odpovídají třídám konkrétního modelu, ale používají se mimo samotné modely, protože z hlediska práce neuronových sítí jsou názvy tříd pouhé označení pro přehlednost a nepoužívají se u práce se surovými daty.

Jak již bylo zmíněno v obecné rovině, továrny na modely jsou strukturovány pomocí návrhového vzoru Abstract Factory, kde každá implementace modelu má svou továrnu a pro daný typ predikce je možné ji vybírat z příkazového řádku.

V této práci existuje několik konkrétních podporovaných typů modelů, konkrétně jde o frameworky Keras, TensorFlow Lite nebo Darknet integrovaný do OpenCV. Pro onen Darknet existuje pouze podpora YOLOv3 modelu, pro ostatní je zde podpora jak detekčního, tak klasifikačního modelu. Pro TensorFlow Lite existuje verze s použitím akcelerátoru EdgeTPU, který však ve finální verzi nebyl použit. V nasazené aplikaci ve finální podobě práce se používají modely v TensorFlow Lite.



Obrázek 5.4: Diagram tříd class selectorů

Class selectory

Součástí rozpoznávací vrstvy jsou tzv. Class selectory, které byly vyčleněny z prediktorů, protože představují obecnou činnost prováděnou v detekci i klasifikaci. V obou případech se totiž vyskytuje výběr třídy podle její konfidence nebo dalších parametrů a o to více bylo potřeba tuto funkcionalitu zobecnit, když existuje více možností výběru třídy nejen podle její konfidence.

Výběr tříd zajišťuje jeden z potomků třídy `ClassSelector`. Všechny implementace sdílejí filtrování tříd podle hraniční konfidence, liší se ve výběru tříd před filtrací. Existují tři implementace:

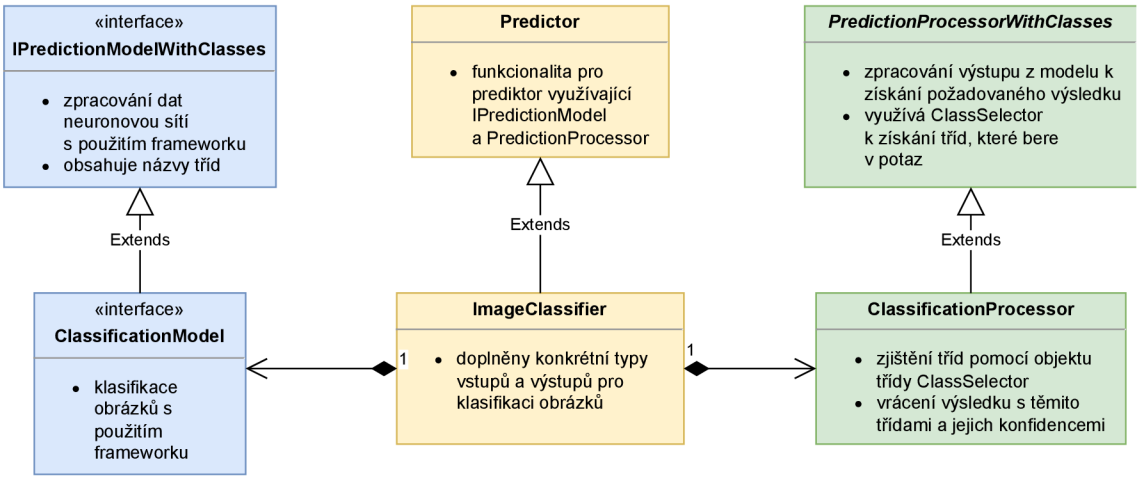
- `FixedClassSelector` –vybere předem definované třídy
- `MaxClassSelector` –vybere třídu s největší konfidencí
- `SortClassSelector` –vybere všechny třídy a seřadí je podle konfidence sestupně

Schéma hierarchie class selectorů je možné nalézt v diagramu 5.4.

Prediktory

Prediktory obalují modely a poskytují nad nimi funkcionalitu, včetně předpřipravení vstupu a získání potřebných dat z výstupu pomocí tzv. prediction processorů. Modely obvykle upravují vstup a výstup pouze, pokud jde o převod týkající se konkrétního frameworku, ale nezpracovávají data v predikci jako celek.

Pro načtení vstupních dat do prediktoru se používají strategie, kde je např. při načítání obrázků ze souboru možné nastavit patřičnou strategii a pak jako vstup modelu dávat místo obrázku cestu k němu. Tato funkcionalita je implementována v abstraktní třídě `APredictor`.



Obrázek 5.5: Diagram tříd klasifikátoru obrázků

Prediktory vycházející z třídy `Predictor` využívají modely pro provedení predikce a následně prediction processory k získání informací z výstupu modelu, jako jsou třídy objektů nebo jejich souřadnice. Prediction processory jsou specifické pro daný druh predikce, ale tok dat přes model a následně přes prediction processor je zde obecný.

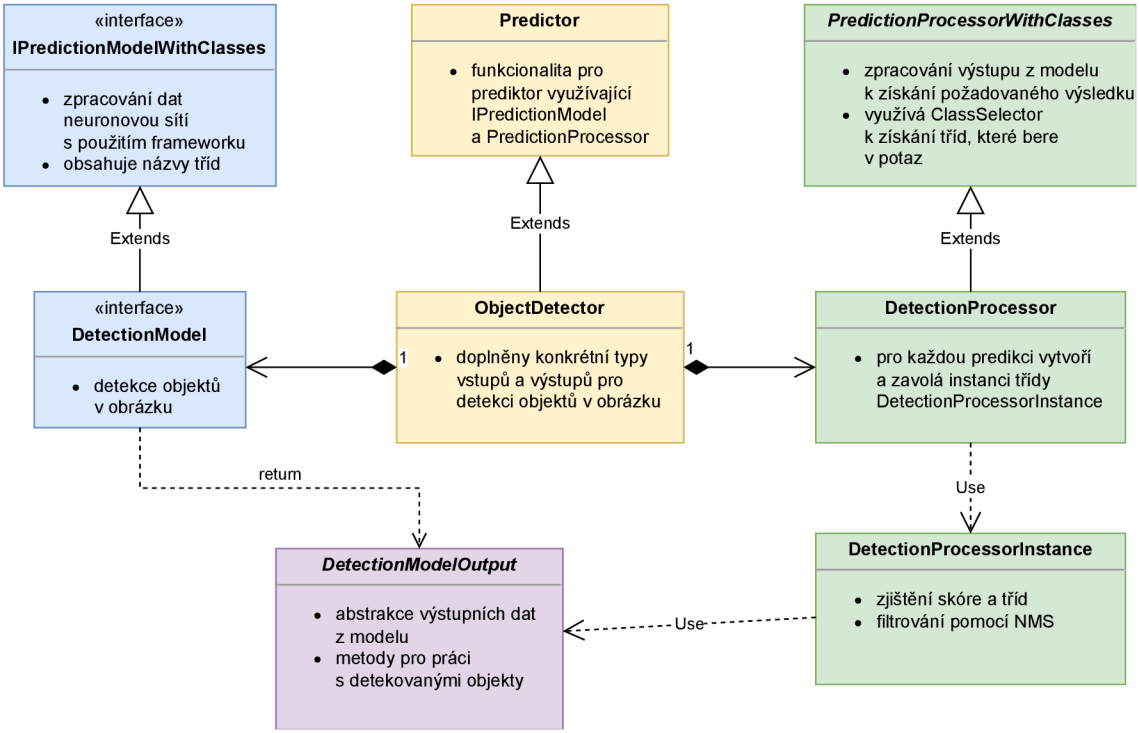
Pro účely aplikace existují specifické typy prediktorů, jsou to `ImageClassifier` pro klasifikaci obrázků a `ObjectDetector` pro detekci objektů v obrázcích. Tyto prediktory využívají kompatibilní modely a disponují specifický prediction processory pro zpracování výstupů.

`ImageClassifier` využívá jednoduchý predictrion processor, protože pouze potřebuje z vektoru výsledků vybrat třídu a vrátit ji a její skóre, na to využívá class selector, viz 5.1.3. Diagram tříd prediktoru `ImageClassifier` je možné nalézt v diagramu 5.5.

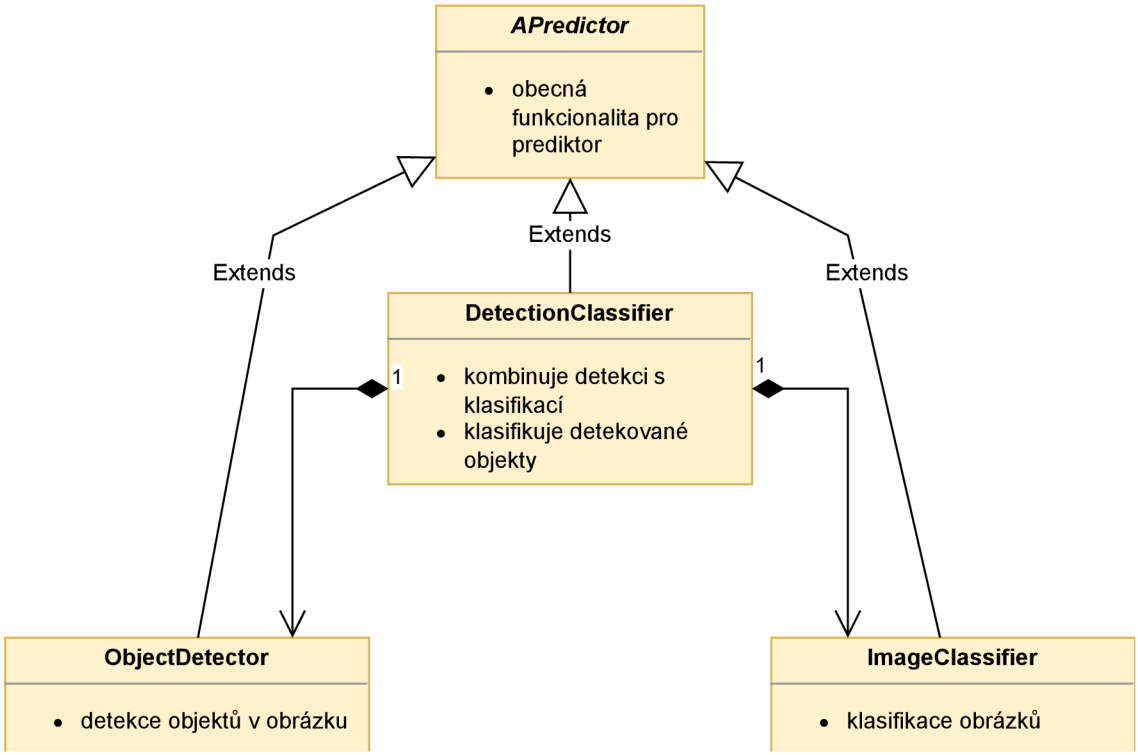
`ObjectDetector` má prediction processor s o něco složitější strukturou, protože pro každou predikci musí zpracovat všechny nalezené objekty, filtrovat je pomocí NMS a pro každý z nich vybírat jejich třídu a konfidenci. I modely jsou složitější a jejich výstup implementuje interface pro získávání jednotlivých detekovaných objektů, které samy implementují jiný potřebný interface, tentokrát pro získání bounding boxu a dále třídy a konfidence, což je opět zajištěno class selectorem. Diagram tříd prediktoru `ObjectDetector` je možné nalézt v diagramu 5.6.

Inicializace prediktorů je zprostředkována továrními objekty, které vytvářejí instance prediktorů a předávají jim potřebné parametry, jako jsou modely a prediction processory.

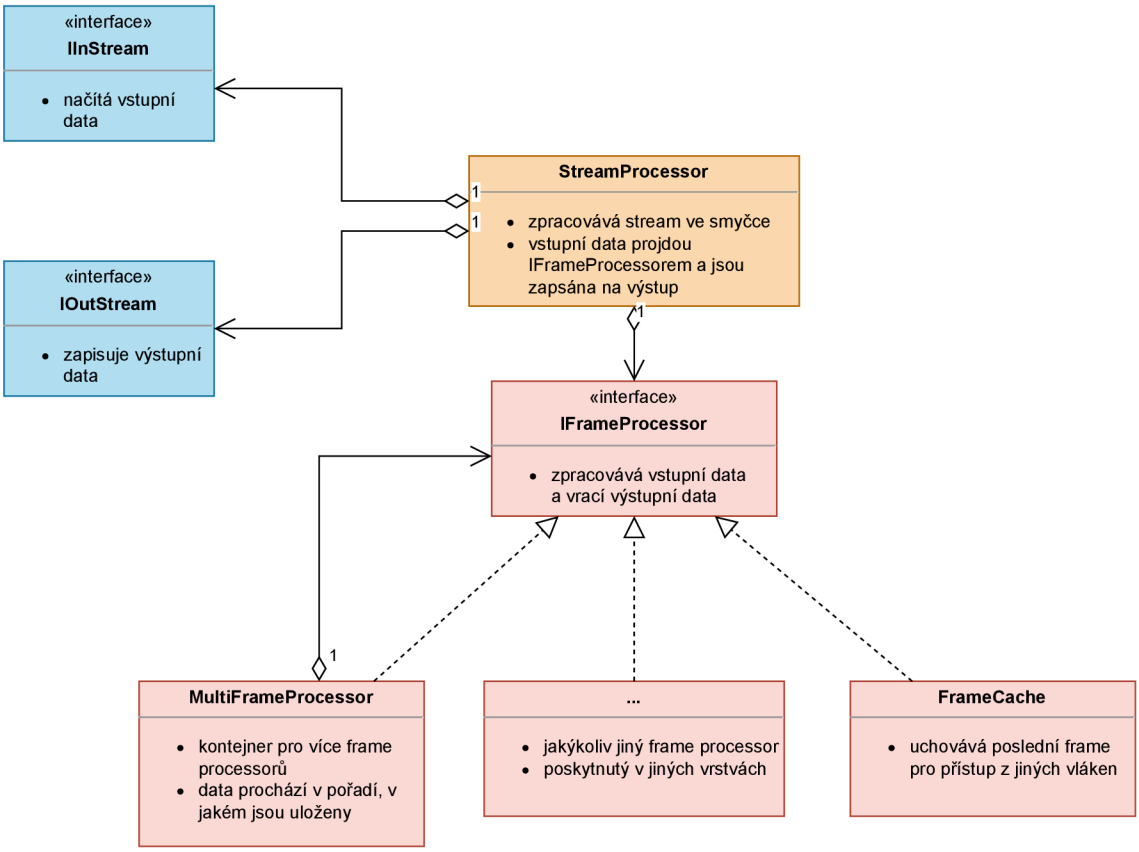
Dále existuje složený prediktor `DetectionClassifier`, který interně využívá jak `ObjectDetector` tak `ImageClassifier`. Je specifický pro danou aplikaci, kde v první fázi se detekují objekty, v této práci ptáci, a poté se nalezené objekty vyříznou a je u nich provedena klasifikace, v této práci klasifikace druhu ptáka. Jelikož jeho funkcionalita plně využívá zmíněné dva prediktory, sám nemá strukturu s modelem a prediction processorem. Jeho diagram můžete nalézt v diagramu 5.7.



Obrázek 5.6: Diagram tříd detektoru objektů



Obrázek 5.7: Diagram tříd klasifikace detekovaných objektů



Obrázek 5.8: Diagram tříd vrstvy streamů

5.1.4 Streamy

Zpracování vstupních a výstupních streamů je řešeno pomocí samostatné vrstvy. Tato vrstva poskytuje prostředky pro vytváření streamů, které jsou vstupní a výstupní, a jejich zpracování na obecné úrovni. Diagram tříd vrstvy vidět v obrázku 5.8

Vstupní streamy implementují rozhraní `IInStream` a poskytují ostatním vrstvám aplikace postupné čtení dat. Výstupní streamy implementují rozhraní `IOutStream` a umožňují postupně data zapisovat. Objekt, který se načítá nebo zapisuje v jednom kroku, se nazývá rámec, nebo anglicky frame.

Frame může být teoreticky libovolného typu, ale v této práci se využívají pouze obrázky reprezentované pomocí vícedimenzionálních polí NumPy, což znamená, že jde o video streamy. K inicializaci těchto video streamů slouží tovární funkce, která vrátí příslušný video stream. Specifická implementace těchto funkcí závisí na konkrétním typu video streamu.

Pro potřeby aplikace, která zpracovává vstupní data a upravená je zapisuje na výstup, jsem vytvořil továrnu na dvojice streamů, která inicializuje vstupní a výstupní stream jako dvojici se stejnými parametry. Tato továrna je parametrická ohledně továren na vstupní a výstupní stream, avšak ve výchozím stavu využívá výchozí továrny.

Implementace video streamů

Vytvořil jsem několik implementací vstupních i výstupních video streamů, odlišné pro živě běžící vlastní aplikaci a pro případné off-line zpracování videosouborů.

Čtení a zápis videosouborů je implementován pomocí streamů, jež interně využívají knihovnu OpenCV. Existuje i implementace výstupních streamů využívající OpenCV v souvislosti s knihovnou GStreamer, od čehož jsem si sliboval, že bude řešit kódování živého streamu, od čehož jsem nakonec ustoupil.

Pro čtení dat z kamery jsem nejprve využíval výše uvedené streamy, které četly data z roury, kam byla data z kamery odesílána z utility libcamera-vid. V současné verzi existuje implementace vstupního video streamu, který přímo ovládá kameru a čte z ní data.

Pro odesílání streamu ve formátu H.264 na RTSP server (zde MediaMTX na stejném stroji), jsem vytvořil implementaci výstupního streamu, který vytvoří podproces FFmpeg, který odesílání zajistí. Kódování využívá hardwarový akcelerátor H.264 v Raspberry Pi. Právě z tohoto důvodu jsem zvolil řešení s FFmpeg, kde jsem vybral patřičný kodek.

Zpracování streamů

Pro zpracování vstupního streamu ve smyčce a odesílání výstupu do výstupního streamu se používá třída `StreamProcessor`, která používá pár vstupních a výstupních streamů, tento pár se při inicializaci vytváří továrnou na dvojice streamů. Kromě toho používá tzv. frame processor pro vlastní zpracování daných framů.

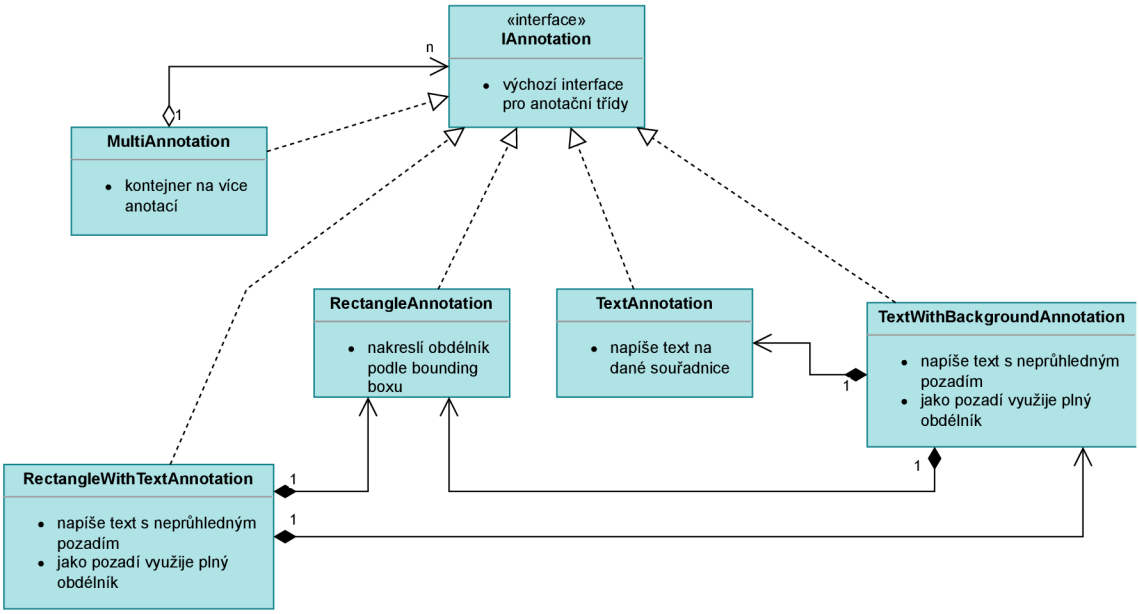
`StreamProcessor` v cyklu načítá z vstupního streamu jeden frame, předá ho frame processoru, který provádí veškeré zpracování dat, a výsledek je zapsán do výstupního streamu. Tato třída zajistí uzavření streamů v případě chyby nebo ukončení cyklu.

Zmíněný frame processor je libovolná třída, která implementuje rozhraní `IFrameProcessor`. Konkrétní frame processory mohou data zpracovávat různými způsoby, například získávat informace nebo upravovat vrácené výsledky. Existují dva obecné frame processory, které se v aplikaci kombinují s další funkcionalitou:

- `FrameCache` – Tento procesor neprovádí žádné úpravy na vstupním framu, ale před jeho vrácením si uloží kopii do paměti, čímž k němu umožňuje přístup z jiných vláken mimo cyklus zpracování streamu.
- `MultiFrameProcessor` – Jedná se o kontejner, který obsahuje kolekci frame processorů. Pro každý frame je každý z těchto procesorů volán řetězově za sebou v pořadí, v jakém jsou v kolekci.

5.1.5 Anotace

Pro aplikaci je klíčové anotovat obrázky ve streamu, což zajišťuje vrstva pro anotaci. Tato vrstva je obecná a umožňuje anotovat libovolný obrázek. Existuje několik



Obrázek 5.9: Diagram anotačních tříd

typů anotací, z nichž každý může obrázek anotovat jiným způsobem (například textem, rámečkem atd.). Anotace využívají knihovnu OpenCV pro kreslení přímo do obrázků.

Instance anotačních tříd anotují obrázky tak, že při jejich vytváření specifikují všechny potřebné atributy, které se liší podle konkrétní třídy anotace. Například třída pro text potřebuje souřadnice, velikost a barvu fontu a samotný textový řetězec, zatímco třída pro rámeček potřebuje rozměry, barvu a tloušťku čáry. Anotace se poté aplikuje na obrázek pomocí příslušné metody, které je předán obrázek, do kterého má být daná anotace zakreslena.

Anotační třídy, jak jsou vidět v diagramu 5.9, jsou následující:

- `MultiAnnotation` – kolekce anotací, které jsou aplikovány na obrázek v určeném pořadí.
- `RectangleAnnotation` – vykreslí obdélník v zadané barvě kolem určeného bounding boxu s možností nastavení tloušťky čáry nebo vyplnění oblasti.
- `TextAnnotation` – zapíše text, který začíná na určeném bodě.
- `TextWithBackgroundAnnotation` – kombinace anotací `RectangleAnnotation` (pro pozadí) a `TextAnnotation` (pro text).
- `RectangleWithTextAnnotation` – kombinace anotací `RectangleAnnotation` (pro rámeček) a `TextWithBackgroundAnnotation` (pro popisek nad rámečkem).

Ukázky jednotlivých anotací je možné nalézt v bakalářském projektu [18].

Obrázek 5.10: Ukázka anotace z výsledků predikce

Anotace streamů

Pro anotaci streamů existuje frame processor `StreamAnnotator`, který uchovává anotaci a aplikuje ji na každý frame, přičemž poskytuje možnost nahrazení této jinou anotací.

5.1.6 Anotace streamů podle predikce

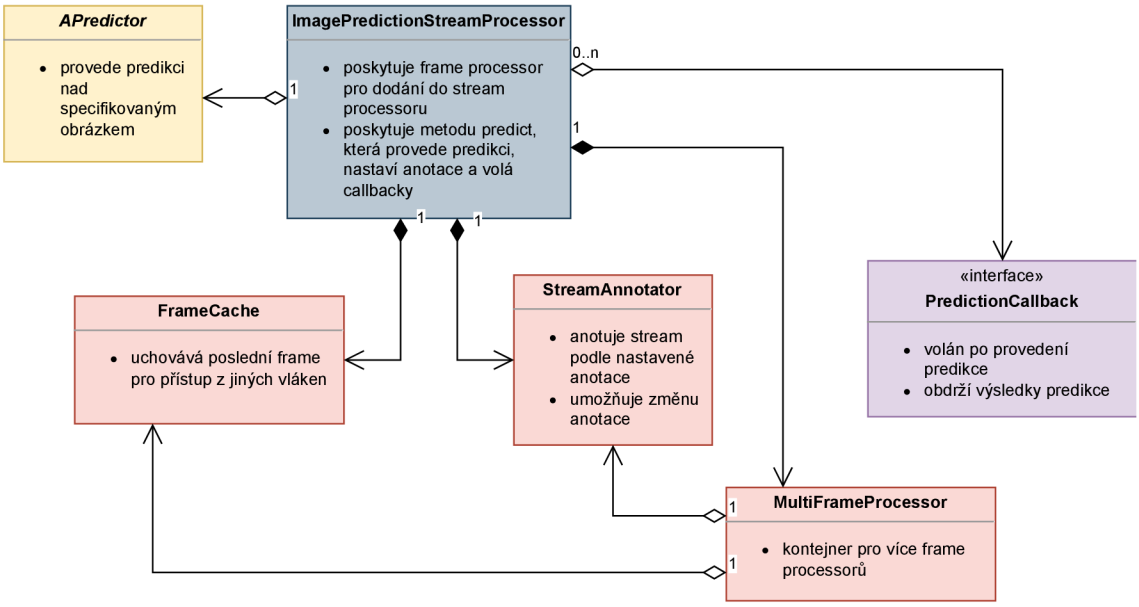
Tato část softwaru téměř tvoří finální aplikaci, která využívá anotace získané z výsledků predikce pro anotaci streamu. Zpracování probíhá periodicky v jiném vlákně, aby se nezdržoval živý stream.

Vrstva je strukturována do tří částí:

- funkce pro vytvoření anotace z predikčních výsledků
- třída poskytující ucelené zpracování
- třídy pro spouštění a řízení běhu zpracování ve vláknech

Vytvoření anotace z výsledků predikce

Funkce `get_prediction_annotation` zpracovává seznam výsledků predikce, který obsahuje bounding box, třídu a její konfidenci. Pro každý výsledek vytváří anotaci typu `RectangleWithTextAnnotation`, více viz 5.1.5, kde rámeček odpovídá souřadnicím bounding boxu a popisný text obsahuje název třídy a procentuální konfidenci. Ukázku anotace je možné vidět v obrázku 5.10.



Obrázek 5.11: Diagram tříd zpracování streamu pomocí predikce

Ucelené zpracování anotace streamů podle predikce

Třída `ImagePredictionStreamProcessor` poskytuje prosředky ke zpracování jednotlivých framů. Obsahuje frame processor `FrameCache` pro uchování posledního framu a `StreamAnnotator` pro anotování podle výsledků poslední predikce. Tyto komponenty jsou spojeny v `MultiFrameProcessor`, který když je předán do `StreamProcessoru`, tak framy procházejí skrz cache a anotátor.

Dále `ImagePredictionStreamProcessor` obsahuje metodu predikce, která je volána z jiného vlákna (viz dále), a spouští predikci, z jejíchž výsledků poté nastaví anotaci a volá callbacky, což umožňuje spouštět libovolný kód po vykonání predikce. Callbacky se využívají v kompletní aplikaci, kde se jako callback dodá sledování a záznam objektů.

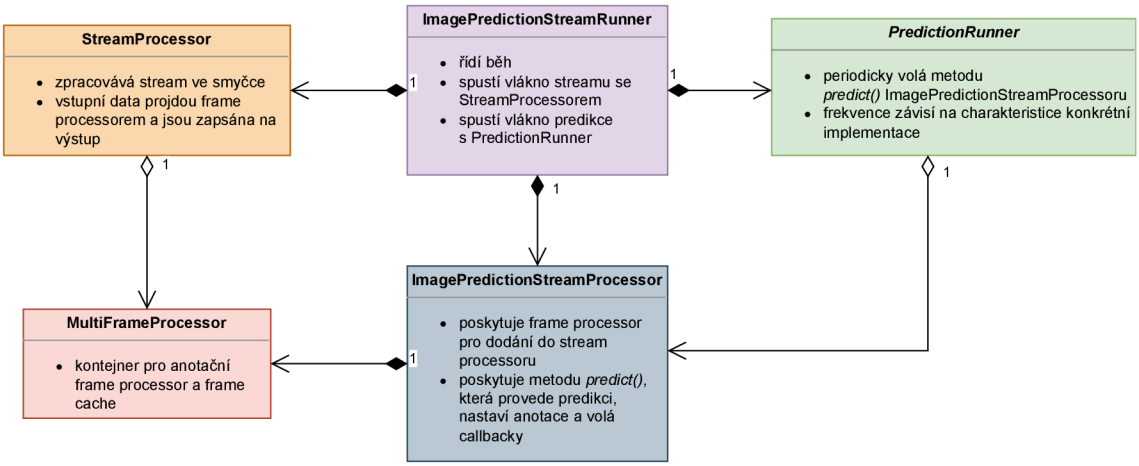
Diagram tříd souvisejících se zpracováním je možné nalézt v diagramu 5.11.

`ImagePredictionStreamProcessor` poskytuje potřebné frame processory a metodu pro provedení predikce a akcí po ní následujících. Nicméně neurčuje, kdy bude metoda predikce volána, ani nespouští `StreamProcessor`. To je úkolem další funkcionality, jak je popsáno v následující sekci.

Spouštění a řízení běhu anotace streamu podle predikce

Jelikož se jedná o jinou činnost, je řízení běhu odděleno od již popsané vlastní logiky. Jak již bylo zmíněno, aplikace běží ve dvou vláknech, kde jedno zpracovává stream rychlostí potřebnou pro plynulé sledování, a druhé provádí predikci, nastavuje anotace a spouští callbacky.

Pro zpracování streamu se používá `StreamProcessor`, jemuž je dodán frame processor z objektu `ÍmagePredictionStreamProcessor`.



Obrázek 5.12: Diagram tříd řízení běhu zpracování streamů podle predikce

Pro spouštění predikce v druhém vláknu slouží `PredictionRunner`, který je abstraktní a připravený na více implementací, které se mohou lišit tím, podle čeho se určí, kdy bude predikce. Tato flexibilita však v aplikaci není využita a používá se jediná implementace, která spouští predikci v intervalu 1 sekundu nebo více, pokud se akce nestihla za tento čas.

K inicializaci `StreamProcessor`u a `PredictionRunner`u a spuštění vláken s nimi se používá třída `ImagePredictionStreamRunner`. Tato se volá z CLI a její metoda běhu čeká na ukončení obou vláken. Diagram 5.12 ukazuje třídy související s řízením běhu.

5.1.7 Sledování a zaznamenávání objektů

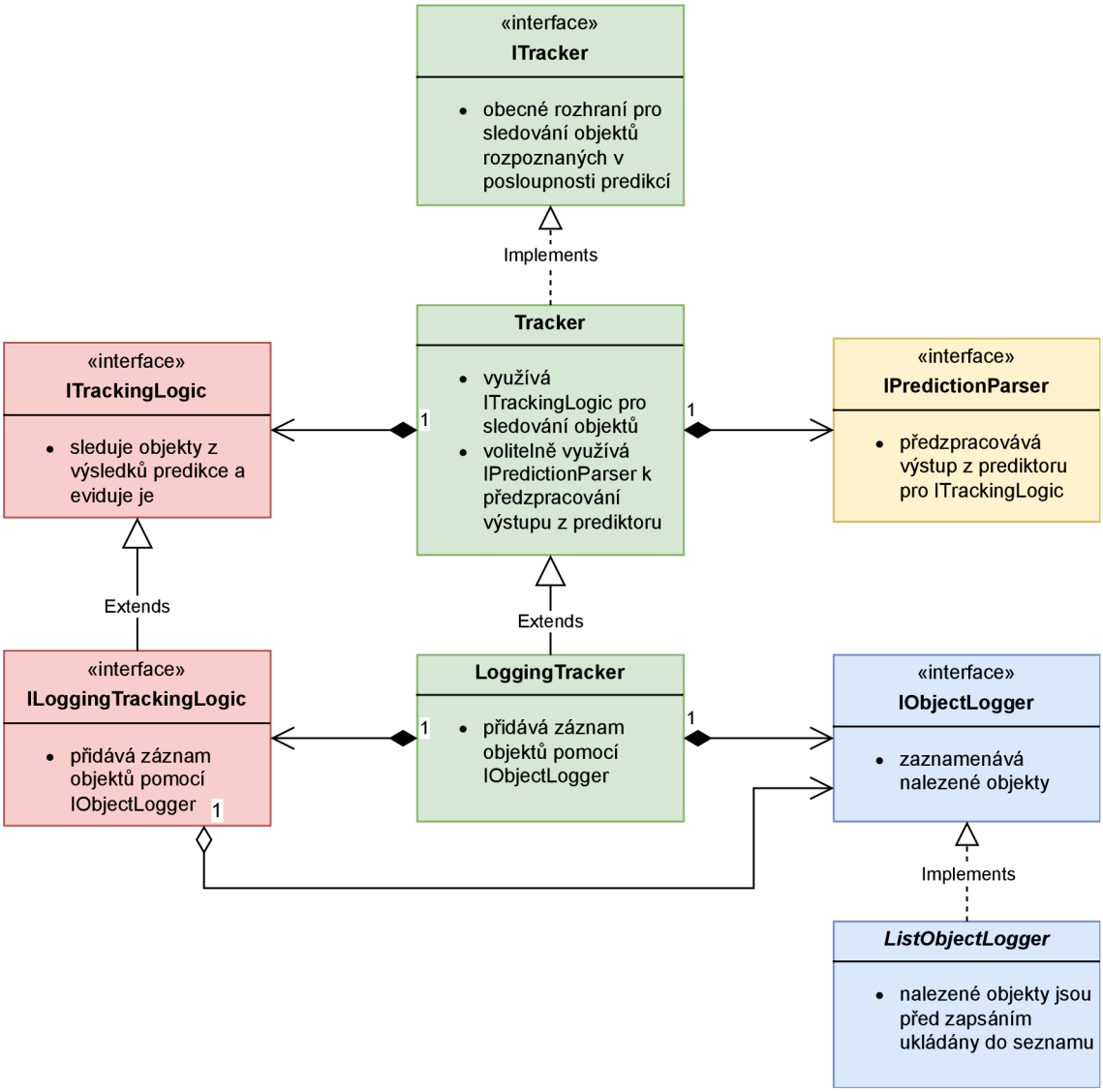
Existuje vrstva, jejíž úloha je sledovat nalezené ptáky a zapisovat je. Ve finální aplikaci jsou data zapisována do databáze, odkud se odesílají do webové aplikace.

Jádrem této vrstvy je třída `Tracker`, která propojuje sledování a objektů a jejich zaznamenávání. Sledování má za úkol propojovat výsledky jednotlivých predikcí a vést si informaci o aktuálně se vyskytujících objektech a má možnost objekty zapsat pomocí druhé části. Diagram 5.13 ukazuje třídy sledování objektů. V něm nejsou vidět jednotlivé implementace, protože je vrstva navržena tak, aby byly vyměnitelné.

Sledování objektů

Pro správné sledování objektů je potřeba propojit jejich nalezení v za sebou následujících framech. K tomu je možné přistoupit různými způsoby a tato vrstva je v tomto ohledu obecná. Pro sledování objektů využívá `Tracker` interface `ITrackingLogic`, jehož implementaci je možné vybrat z příkazového řádku.

V případě bezchybné predikce a jejího rychlého zpracování by bylo možné použít algoritmus SORT, v případě mé práce jsem použil vlastní algoritmus, který je také jedinou implementací `ITrackingLogic`, která se nazývá `IntervalTrackingLogic`.



Obrázek 5.13: Diagram tříd sledování objektů

Tento algoritmus se nezabývá souřadnicemi objektů, ale pouze jejich výskytem v čase. Pro každou třídu zvlášť eviduje nalezené objekty a to tak, že si nově nalezený objekt pamatuje a za totožný jej pokládá, pokud je viděn v určitém intervalu od doby posledního nalezení. Po uplynutí intervalu je objekt zapsán a smazán z evidence. V případě více objektů stejné třídy eviduje více takových objektů, pro každý počet jeden, kde objekt příslušící k určitému počtu se považuje za znovu viděný pouze tehdy, pokud je opět viděn daný počet objektů nebo vyšší. Vzhledem k nedokonalosti rozpoznávání je však také pro objekt každého počtu zvýšen interval, ve kterém je považován za totožný.

Z uvedeného algoritmu vyplývá, že se nezaznamenává reálný počet ptáků, ale počet jejich výskytů, který je mnohonásobně vyšší než odhadovaný počet jedinců.

Zaznamenaný objekt třídy `IntervalTrackingLogic` je typu `ClassLoggedObject`, který obsahuje název třídy (druhu ptáka) a časy příchodu a odchodu.

Zaznamenávání objektů

V závislosti na použité logice sledování jsou v určité chvíli objekty zaznamenávány. K tomu `Tracker` používá interface `IObjectLogger`, jehož implementaci je opět možné vybrat z příkazového řádku.

V práci existují dvě implementace `IObjectLogger`. Výchozí implementace, `FileObjectLogger` zapisuje do textového souboru nebo na standardní výstup tak, že vypíše řetězcovou reprezentaci daných objektů pomocí funkce `pprint` z knihovny jazyka Python. Druhá implementace je dodána vrstvou pro databázi zaznamenaných objektů, více 5.1.8.

5.1.8 Databáze zaznamenaných objektů

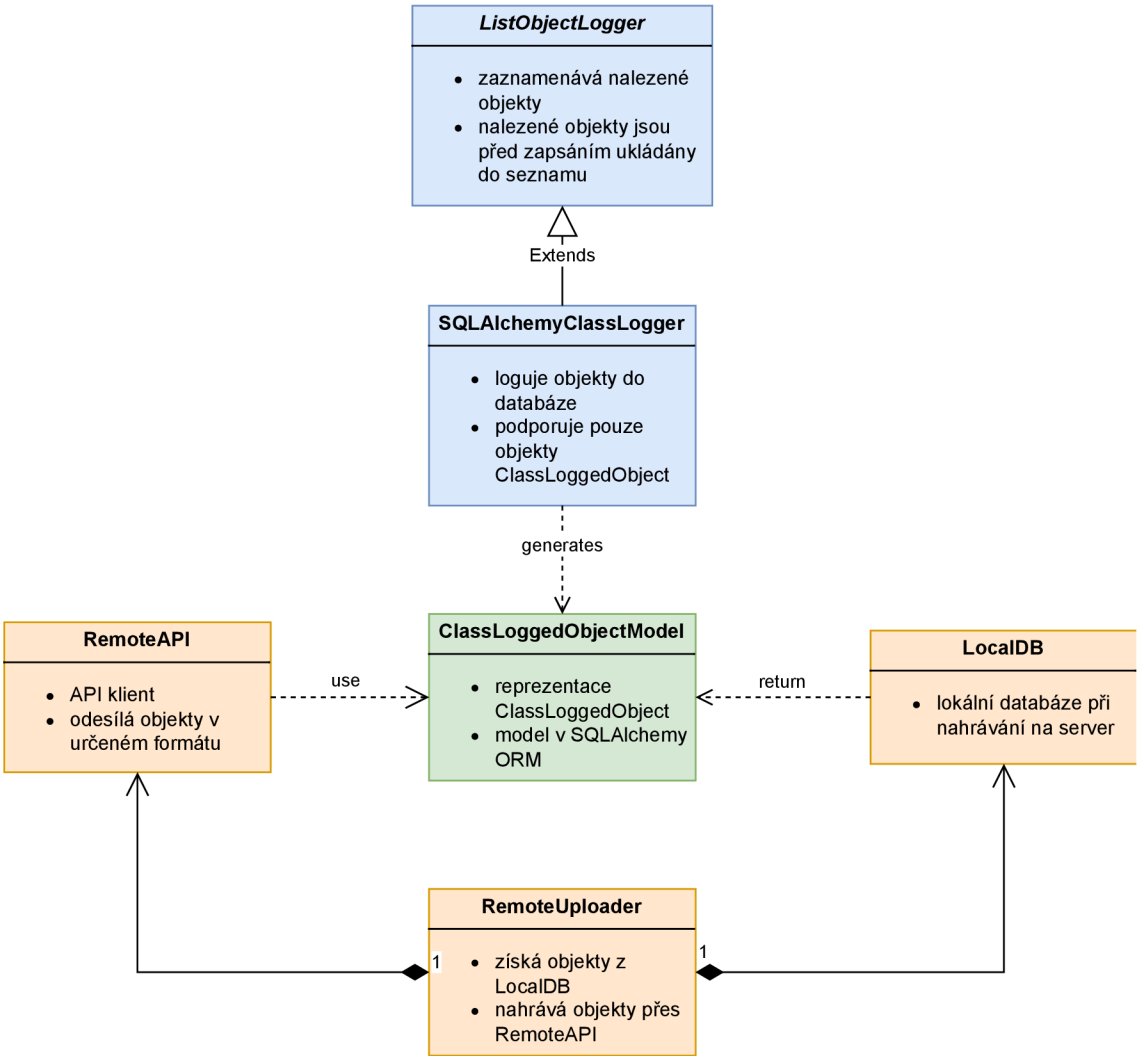
Vrstva pro databázi zaznamenaných objektů definuje databázovou strukturu a umožňuje zapisování objektů do ní a zároveň odesílání databáze do webové aplikace pomocí jejího API, což je však prováděno mimo běh aplikace, voláním samostatného vstupního bodu z CLI. Diagram 5.14 ukazuje třídy databáze zaznamenaných objektů..

Přístup k databázi je řešen pomocí knihovny SQLAlchemy, na vlastní implementaci databáze tedy tolik nezáleží, pokud je podporovaná touto knihovnou. Databázové schéma je inicializováno pokud na daném URL neexistuje. V práci jako databázi používám SQLite, protože není potřeba databázový server.

Pomocí výše zmíněné knihovny je definován databázový model obsahující jednu tabulku `LoggedObjectModel`, která obsahuje třídu, čas příchodu a odchodu ptáka a ID v databázi webové aplikace, což se používá při nahrávání, více v dalších odstavcích.

Zaznamenávání sledovaných objektů do databáze

Pro zapisování objektů do databáze vrstva definuje třídu `SQLAlchemyClassLogger`, implementaci rozhraní `IObjectLogger`.



Obrázek 5.14: Diagram tříd databáze zaznamenaných objektů

Nahrávání zaznamenaných objektů do webové aplikace

Pro nahrávání do webové aplikace slouží vstupní bod v CLI, kterému je kromě URL lokální databáze předáno URL vedoucí na API webové aplikace a uživatelské jméno a heslo k němu.

Pro nahrání se vybírají pouze objekty, jejichž hodnota sloupce `remote id` je `NULL`, což značí, že ještě nebyly nahrány do webové aplikace. Požadavek volá u API endpoint `upload`, kam ve formátu JSON odešle seznam objektů, obsahujících třídu, čas příchodu a čas odchodu.

API vrací seznam identifikátorů, které byly přiřazeny odpovídajícím objektům v databázi webové aplikace. Ty jsou do lokální databáze zapsány do speciálního sloupce v tabulce, čímž je docíleno, že se nebudou znovu nahrávat.

5.1.9 Vrstva kompletní aplikace

Kompletní aplikace je, kromě jejího vstupního bodu do CLI, definována třídou `App`. Tato třída používá pro svou činnost dva objekty, které dá do souvislosti.

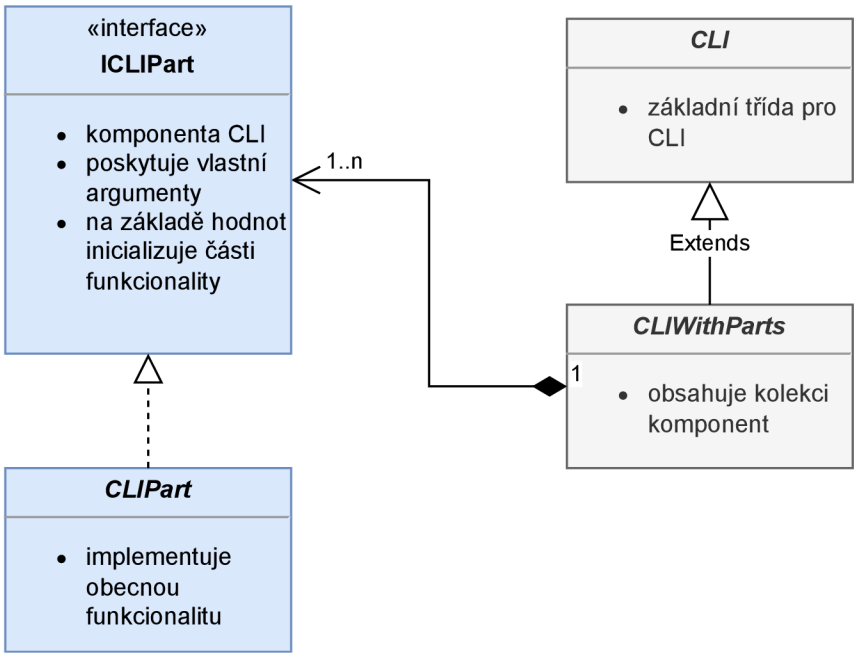
Hlavním takovým objektem je instance třídy `ImagePredictionStreamProcessor`, která, jak již bylo uvedeno, zpracovává stream tak, že u něj provádí rozpoznávání obrazu a s jeho výsledky anotuje stream. Další část je `DatabaseObjectLogger`, jehož patřičná metoda je přidána jako callback do instance třídy `ImagePredictionStreamProcessor`.

Výše uvedeným způsobem je zajištěna kompletní funkcionalita aplikace na úrovni objektového kódu, který je volán z vrstvy rozhraní příkazového řádku, odkud mu jsou také dodány prediktor, sledovač objektů a streamy, jak je uvedeno v sekci 5.1.10. Důvod, proč se inicializují v CLI samostatně, je, protože každý z nich má vlastní sadu parametrů pro CLI, kterými je možné ovlivňovat jejich vytváření.

5.1.10 Rozhraní příkazového řádku

Jak již bylo zmíněno, rozhraní příkazového řádku (CLI) je vstupním bodem při spouštění a slouží k inicializaci objektové struktury a následnému spuštění vlastní funkcionality. Vstupní body existují kromě celkové aplikace také pro některou dílčí funkcionalitu, kterou lze spouštět samostatně. Konkrétně existují následující vstupní body:

- kompletní aplikace
- detekce objektů v obrázku
- klasifikace obrázků
- klasifikace detekovaných objektů
- anotace streamu pomocí detekce
- anotace streamu pomocí klasifikace detekovaných objektů



Obrázek 5.15: Obecný diagram tříd CLI

CLI je zajištěno samostatnou vrstvou, která poskytuje obecné třídy společné pro všechny vstupní body, dále obecné implementace pro inicializaci jednotlivých vrstev a implementace vstupních bodů, jež využívají zmíněné prostředky.

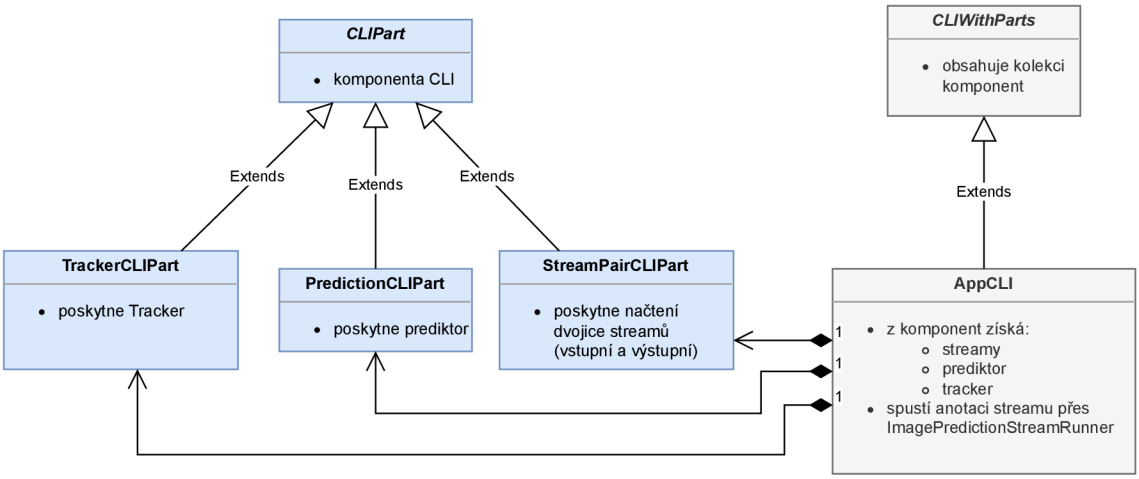
Hlavní funkcí této vrstvy je zpracování parametrů z příkazového řádku, jejichž hodnoty buď přímo, nebo nepřímo slouží jako parametry do továrních objektů nebo funkcí, které inicializují objekty vlastní funkcionality. Po inicializaci tato vrstva spustí funkcionalitu a v případě některých vstupních bodů vypíše výsledek funkcionality do konzole. Diagram 5.15 ukazuje třídy této vrstvy.

Pro samotné zpracování parametrů z příkazového řádku se používá vestavěná knihovna `argparse`. Vzhledem k tomu, že některé parametry jsou uloženy v souboru, povolil jsem možnost načítání argumentů ze souboru použitím znaku `@`.

Pro skládání CLI z různých znovupoužitelných částí jsem vytvořil rozhraní, které umožňuje implementovat komponenty CLI, které samy o sobě nespouštějí funkcionalitu, ale definují potřebné parametry a poskytují možnost vytvoření objektů z hodnot těchto parametrů. Pro samotný vstupní bod pak stačí tyto komponenty definovat před zpracováním parametrů a následně je použít k inicializaci objektů pro vlastní funkcionalitu.

Vstupní bod kompletní aplikace

Vstupní bod aplikace využívá pro svoji funkčnost několik CLI komponent. Jedná se o CLI komponentu klasifikátoru detekcí, který sám využívá dvě instance CLI komponenty predikce, a to pro detektor a pro klasifikátor. Dále používá komponentu pro získání páru streamů, která umožňuje volit typ vstupního a výstupního streamu. Pro sledování a záznam objektů používá také specifickou CLI komponentu.



Obrázek 5.16: Diagram tříd vstupního bodu CLI aplikace

Z daných komponent je poskládán objekt aplikace, jehož funkcionalita je následně spuštěna.

Schéma tříd v případě vstupního bodu kompletní aplikace je možné nalézt v diagramu 5.16.

5.1.11 Prostředí napsané v shellu

Jak již bylo uvedeno, CLI je vstupním bodem nejen do aplikace. Objektová struktura je inicializována z hodnot parametrů. CLI však v případě této práce neřeší záležitosti, které jsou mimo samotný kód v Pythonu a parametry i s jejich výchozími hodnotami jsou ještě příliš obecné a pro spuštění funkční aplikace by bylo do příkazového řádku potřeba uvádět mnoho parametrů, které se mohou lišit dle aktuálního prostředí, někdy je také nutné spouštět externí programy.

Pro zavedení určitého prostředí okolo kódu v Pythonu jsem vytvořil obalovou vrstvu napsanou v shellu, konkrétně bashi. Ta umožňuje načítání proměnných a spouštění nejen aplikace s parametry nich anebo společně se spuštěním externích programů. Tato vrstva také aktivuje virtuální prostředí Pythonu a nastaví správné cesty pro hledání modulů tohoto jazyka.

Proměnné jsou načítány ze souboru evidovaného ve verzovacím systému a poté z lokálního souboru. Dále jsou proměnné načítány z různých souborů v závislosti na tom, jestli se jedná o vývojové nebo ostré prostředí. To umožňuje mít větší variabilitu prostředí.

V projektu existuje několik shell skriptů, které spouštějí nějaký vstupní bod CLI v kontextu tohoto prostředí. V případě aplikace shell skript kromě aplikace spustí i program MediaMTX a pro aplikaci zvolí takové parametry, aby byl výstupní stream odesílán právě do něj. Kromě toho definuje několik dalších parametrů.

Prostředí může být aktivováno v interaktivním shellu, což může být někdy potřeba. Pro spouštění kompletní funkcionality přes shell skripty to však není nutné, protože tyto skripty si samy pro sebe toto prostředí aktivují.

5.2 Webová aplikace

Pro zobrazení streamu z kamery a zobrazování výsledků jsem vyvinul webovou aplikaci, která je také napsaná v jazyce Python 3 a využívá framework Django.

Aplikace má dva pohledy, anglicky zvané views, kde v jednom se zobrazují statistiky ve formě grafů, a v druhém je možné zobrazit živý stream. Je podporováno přihlášení uživatelů, které se používá pro omezení přístupu k živému streamu, protože na něm mohou být vidět lidé pobývající na chalupě. Uživatelské rozhraní je dostupné v angličtině a češtině.

5.2.1 Obecná funkcionalita webové aplikace

Jak již bylo zmíněno, webová aplikace je postavena na frameworku Django, ze kterého využívá databázi, přihlašování uživatelů a lokalizační systém.

Webový server

Jako webový server jsem použil mod_wsgi-express, což představuje separátní instanci Apache nakonfigurovanou tak, že používá modul mod_wsgi z virtuálního prostředí webové aplikace.

Tato instance poskytuje veškeré prostředky pro webovou aplikaci kromě finálního zpřístupnění do Internetu přes HTTPS, to je úkolem hlavní instance webového serveru Apache, kde je webová aplikace přístupná pod jedním z virtualhostů.

Přístup s mod_wsgi-express jsem zvolil z důvodu, že jsem kvůli streamu potřeboval autorizaci k cestám na úrovni webového serveru pomocí autentizace z Djanga. Proto jsem nemohl na Django použít webový server dedikovaný na WSGI a ostatní cesty poskytovat přímo z hlavní instance Apache. Na druhou stranu jsem nemohl použít mod_wsgi instalovaný do systémové instance Apache, jelikož verze dané linuxové distribuce primárně používá příliš starou verzi Pythonu, konkrétně 3.8.

Webdesign

Pro webovou aplikaci jsem vytvořil vlastní design v podobě šablon a statických CSS souborů. Šablony jsou napsány v šablonovacím jazyce frameworku Django, HTML se generuje na serveru. JavaScript se užívá pouze pro specifické akce na straně klienta.

Základem designu je hlavní šablona, která definuje základní strukturu stránky. Tuto šablonu rozšiřují šablony jednotlivých views, které do patřičných bloků doplní vlastní obsah. Kromě bloku na hlavní obsah, který je v HTML elementu `<main>` jsou definovány např. bloky pro vložení dalšího CSS a JavaScriptu.

Tento design podporuje tzv. sezónní barvy, což znamená, že barevný nádech celého webu se mění v závislosti na ročním období. To je na straně serveru řešeno dynamickým vložením odkazu na CSS soubor, který doplní styly o tyto barvy.

Obrázek 5.17: Přihlášení do webové aplikace

Lokalizace

Jak již bylo zmíněno, používá se lokalizační systém frameworku Django, který pro samotný překlad používá systém gettext. Hlášky aplikace jsou psané v angličtině a je k dispozici překlad do češtiny.

Pro definici překládaných hlášek se používá soubor `.po`, který je pro nasazení nebo testování potřeba zkompilovat do souboru `.mo`.

Výběr jazyka je primárně ponechán na systému Django, který se rozhoduje podle uživatelského jazyka. Z webové aplikace je možné implicitně změnit jazyk, k čemuž jsou určeny ikonky představující vlaječky, konkrétně vlaječka Spojeného království pro angličtinu a vlaječka České republiky pro češtinu.

Přihlášení

Z důvodů ochrany soukromí jsem chtěl omezit přístup ke streamu na uživatele s patřičným oprávněním, k čemuž jsem využil vestavěný přihlašovací systém frameworku Django.

Kvůli zabezpečení streamu jsem na instanci webového serveru mod_wsgi-express implementoval autorizaci na úrovni webového serveru, která je napojena na přihlašovací systém Djanga, takže pro uživatele je toto transparentní.

Pro přihlašovací formuláře používám vlastní views, které využívají design webu. Vzhled přihlašovacího view je vidět ve screenshotu 5.17.

Přihlašování je navrženo tak, aby každá osoba měla vlastní uživatelský účet. Pro první přihlášení je uživateli zaslán uvítací email, který vyzývá k nastavení hesla Lidé si heslo také sami resetují heslo pomocí podobné emailové komunikace. Uživatel je tedy naprosto samostatný vzhledem k zapomenutým heslům.

5.2.2 Statistika

Pro zobrazování statistiky jsem vytvořil view, který obsahuje graf zobrazující výskyt jednotlivých druhů ptáků v určitých časových obdobích. Pro nahrávání existuje webové API.

API pro nahrávání zaznamenaných objektů

API je realizováno pomocí Django REST Framework a umožňuje nahrání zaznamenaných objektů z rozpoznávací aplikace do místní databáze webové aplikace.

Zabezpečení je provedeno přihlášením do systému Django, tentokrát pomocí HTTP Basic auth, které je pomocí django rest framework, napojeno na přihlašovací systém Djanga. Pro autorizaci je potřeba, aby uživatel měl oprávnění přidávat zaznamenané objekty do databáze. V nasazení pro to používám speciální uživatelský účet.

API obsahuje endpoint, který pomocí JSONu přijme seznam objektů, kde každý objekt obsahuje název třídy a časy příchodu a odchodu. Endpoint zapíše objekty do vlastní tabulky a vrátí ID nově vytvořených záznamů.

View pro zobrazení statistiky

Pro zobrazení statistiky z dat odeslaných do webové aplikace jsem vytvořil view, který zobrazuje data ve sloupcovém grafu, kde pro každou časovou jednotku jsou nad sebou umístěné barevné sloupce znázorňující počet výskytů ptáka daného druhu.

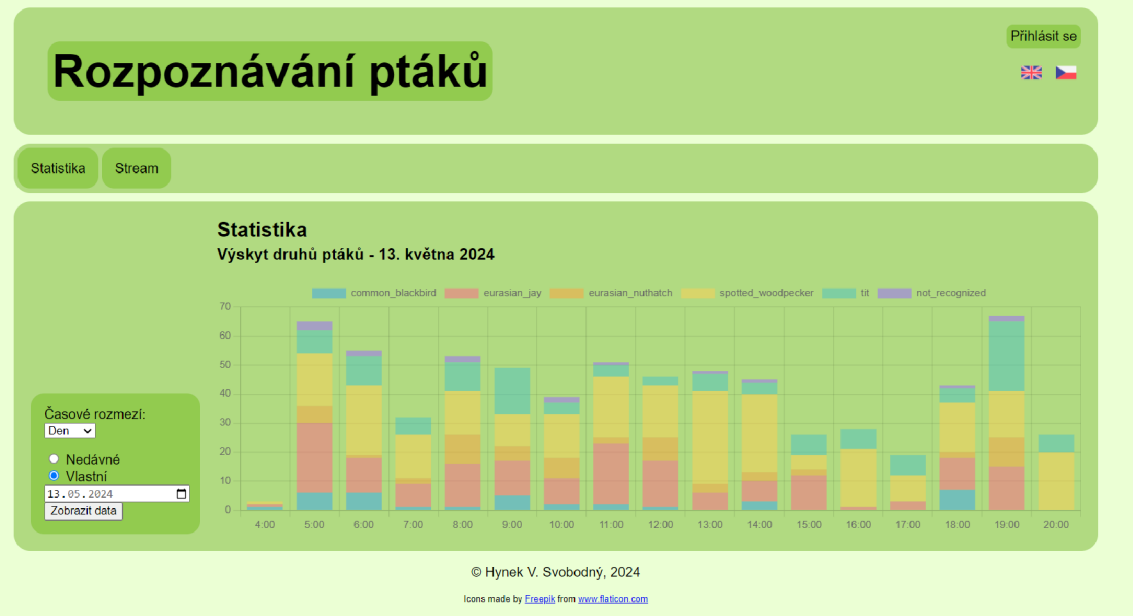
View je flexibilní ohledně výběru délky časového období a času za který se statistiky zobrazí. Délka časového období určuje rozsah daného zobrazení a velikost časové jednotky, do které se budou data agregovat. Na výběr jsou následující, uvedeno rozsah (časová jednotka agregace):

- rok (měsíc)
- měsíc (den)
- týden (den)
- den (hodina)

Volba je umožněna ve formuláři umístěném v levé části view.
Screenshot view statistiky je možné vidět na obrázku 5.18.

5.2.3 Živý stream

Živý stream je k dispozici na chráněném view, který je přístupný pouze přihlášeným uživatelům ve skupině `stream_viewers`. Samotný stream se nepřenáší přes Django, a k jeho ochraně bylo nutné použít zabezpečení na úrovni webového serveru.



Obrázek 5.18: View pro zobrazení statistiky

Zpřístupnění streamu

Stream je poskytován instancí programu MediaMTX, určenou pouze pro webovou aplikaci. Tato se v případě, že klient začne sledovat stream, připojí na instanci běžící v rámci rozpoznávací aplikace na Raspberry Pi. Díky tomu se z lokální sítě na chalupě přenášení data pouze jednou v případě jakéhokoliv počtu kientů, nebo vůbec, pokud nikdo stream nesleduje, což vede k menšímu zatížení tamní lokální sítě.

Instance MediaMTX na serveru zpřístupňuje stream pomocí protokoku HLS. Tento je pak klientům zpřístupněn ve webovém serveru mod-wsgi přes reverzní proxy. Cesta k němu je omezena na skupinu uživatelů `stream_viewers` díky autorizaci na úrovni webového serveru napojené na přihlašování frameworku Django.

View živého streamu

Pro zobrazení streamu v rámci webu jsem vytvořil view, který využívá šablonu webu, ve které do hlavního bloku vloží živý stream.

Samotný element živého streamu je ovládán knihovnou Video.js, kterou jsem použil pro umožnění plynulého živého streamu.

View je, stejně jako přímý stream, omezen na patřičné uživatele. V tomto případě však již postačí autorizace přes Django.

Vzhled view streamu je možné vidět v screenshotu 5.19.

Obrázek 5.19: View živého streamu

6 Modely neuronových sítí

Jak již bylo zmíněno, rozpoznávání je rozděleno na dvě části. První je detekce objektů v obrázku a druhá klasifikace obrázků. Pro obě tyto úlohy je potřeba mít vhodný model. Model pro detekci objektů v obrázku jsem použil předtrénovaný, zatímco model pro klasifikaci jsem si natrénoval pomocí transfer learning.

6.1 Sběr dat

Pro účely trénování anebo evaluace jsem sbíral množství dat na krmítku. Pro sběr jsem původně využíval fotopast, jak je zmíněno v projektu. Dále jsem používal motionEye, který ptáky snímal na základě detekce pohybu.

Pro motionEye jsem musel upravit získávání dat z kamery, aby fungovalo zaostřování s v té době používanou kamerou Arducam 16Mpx. To způsobovalo náhlá selhání nebo neschopnost kamery se inicializovat. Přesto jsem však v určitém období nasbíral dostatečné množství dat.

Z fotopasti jsem nasbíral data, která jsem rozdělil do tří dávek, kde každá dávka obsahuje několik adresářů podle dne pořízení dat. Pořizovány byly pouze fotografie. V případě dat z motionEye byla data sbírána ve formě videosouborů, ze kterých jsem potom manuálně extrahoval fotografie, přičemž jsem neextrahoval všechny snímky z videí, protože by často představovaly podobná data a zabíraly by mnoho úložiště, nýbrž jsem extrahoval pouze vybrané snímky.

6.2 Model pro detekci objektů

Pro detekci objektů jsem v práci použil předtrénovaný model YOLO, nejprve jsem používal verzi YOLOv3, konkrétně implementaci Darknet integrovanou v OpenCV, experimentoval jsem i s verzí YOLOv4. Ve finální verzi práce jsem použil YOLOv8 od Ultralytics, který jsem exportoval do TF Lite pomocí jejich nástroje [2].

Zamýšlel jsem i trénování vlastního modelu, od čehož jsem ustoupil, protože natrénovaný model nepřinášel žádné rozumné výsledky, ani jsem ho tedy netestoval. Domnívám se, že důvod neúspěchu byl nedostatek trénovacích dat.

6.2.1 Validace modelu pro detekci objektů

Přestože jsem netrénoval vlastní model, evaluoval jsem ho na vlastních anotovaných datech. Data, která jsem použil pro evaluaci, jsou anotované fotografie extrahované

z videí z motionEye. Data pocházejí z asi tří dní.

Pro evaluaci jsem opět použil nástroj od Ultralytics, pro který jsem vytvořil definici datasetu, aby data našel.

Při evaluaci na vlastních anotovaných datech model dosáhl mAP50 skóre 74,8 % a mAP50-95 skóre 43 %.

6.3 Model pro klasifikaci obrázků

Pro klasifikaci jednotlivých druhů ptáků jsem vyvinul vlastní model s názvem "jiz-birds", který je zaměřen na ptáky v Jizerských horách, kde se nachází krmítko. Byl natrénován pomocí transfer learning z modelu EfficientNet natrénovaného na datasetu ImageNet. Existují dvě verze modelu.

6.3.1 Datasety

Pro první verzi modelu jsem použil jednu dávku z fotopasti, na testování ostatní dávky. Nechal jsem extrahovat obrázky ptáků pomocí detekčního modelu, v té době YOLOv3, kterou jsem poté ručně rozřadil.

Ve druhé verzi jsem na trénování použil všechny dávky z fotopasti a na testování jednu dávku z dat z motionEye. Data jsem již nechal předrozdělit první verzí modelu, potom jsem data ručně zkontroloval a opravil chybná zařazení. Oproti první verzi jsem přejmenoval některé názvy tříd, pokud lépe vystihují více příbuzných druhů.

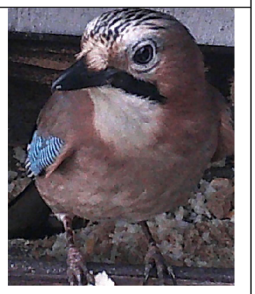
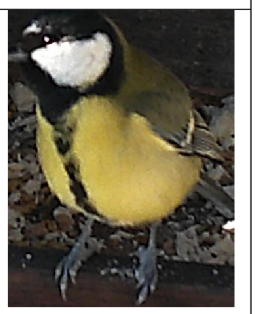
V obou případech jsem z fotografií odstranil ty, které byly pořízeny nočním viděním fotopasti, což kazí barvy. Dále také byly občas identifikovány jiné objekty jako pták, z nichž zajímavé výsledky můžete nalézt v souvisejícím bakalářském projektu [18], jednalo se například o člověka nebo lavičku. Tyto bylo také potřeba odstranit.

6.3.2 Trénovací skript

Trénovací skript není strukturován objektově jako hlavní část softwaru, ale je napsán jako procedurální kód, který postupně načítá data a provádí všechny operace spojené s trénováním. Data určená k tréninku byla automaticky rozdělena na trénovací a validační sady pomocí prostředků trénovací knihovny. Přesné cesty k souborům, další parametry a přesměrování výstupu do souborů jsou nastaveny při spouštění skriptu v shellu. Při tvorbě tohoto skriptu jsem se v mnohém řídil návodem z oficiální dokumentace frameworku Keras [10], ve kterém jsem také model trénoval.

6.3.3 Trénování

Model jsem natrénoval pomocí trénovacího skriptu. Popis datasetu, který jsem použil pro trénování verze v0.2 je možné vidět v tabulce.

Druh(y) ptáka	**Název třídy**	**Počet obrázků**	**Ukázkový obrázek**
kos černý	`common_blackbird`	14	
sojka obecná	`eurasian_jay`	1260	
brhlík lesní	`eurasian_nuthatch`	269	
strakapoud (velký, malý, prostřední)	`spotted_woodpecker`	290	
sýkora (koňadra, modřinka, babka)	`tit`	361	

Tabulka 6.1: Trénovací sada jizbirds-v0.2

Název třídy	Počet obrázků	Počet chyb	Přesnost na datech
common_blackbird	5	0	100,0 %
eurasian_jay	125	14	88,80 %
eurasian_nuthatch	42	15	64,29 %
spotted_woodpecker	14	2	85,71 %
tit	281	4	98,58 %
Celkem	**467**	**35**	**92,51 %**

Tabulka 6.2: Testování modelu jizbirds-v0.2

6.3.4 Testování

Model jsem otestoval na jiné datové sadě. Podrobnosti testu jsou k nalezení v tabulce.

7 Výsledky

Počínaje nasazením rozpoznávací i webové aplikace probíhá nepřetržité sledování krmítka v reálném čase a záznam výsledků. Data jsou dostupná v od 5. 5. a v této práci je statistika uvedena ke dni 14. 5. 2024. Jsou tedy dostupná data z necelých 10 dní.

Základním zjištěním, které bylo do značné míry zřejmé z povahy kamery a chování ptáků je, že ptáci byli zaznamenáni pouze za světla, v tuto dobu je to konkrétně od 5. hodiny ráno do 8. hodiny večer včetně. V několika dnech se vyskytly ojedinělé případy před 5. hodinou. Výskyty před 5. hodinou jsou v pozdějších dnech, což připisuji posouvání doby svítání.

Co se týká výskytu jednotlivých druhů, nejmenší počet zaznamenání měl kos černý. Následuje brhlík lesní, sýkora. Strakapoud a sojka obecná jsou nejčastějšími návštěvníky. Toto pořadí se vcelku nemění. Domnívám se, že to je dáno různým počtem jedinců u různých druhů.

Pokoušel jsem se určit trendy, v jaké denní době se ten který druh vyskytuje více a kdy méně, ale zjistil jsem, že žádné takové trendy neexistují. Existuje však trend celkových výskytů v každém dni a spočívá v tom, že v brzkých ranních hodinách se vždy vyskytuje víde ptáků, než přes poledne. Večerní výskyt je daný podle intenzity denního světla.

Jeden z trendů, který jsem vypozoroval byl však to, že celkový denní výskyt přibýval. Domnívám se, že důvodem je hnízdění ptáků a nutnost dokrmovat se častěji.

Výsledky byly částečně nepřesné díky druhům ptáků, na které nebyl systém natrénován a kteří navštívili krmítko v podstatě náhodně. Jednalo se například o pěnkavu. Tito ptáci byli zařazeni do různých druhů, protože konfidence byla vysoká na to, aby se dal druh označit jako nerozpoznaný.

Dalším důvodem nepřesnosti byl výskyt veverky, která byla v některých případech identifikována jako pták, ale naštěstí ne vždy. Konfidence také způsobila, že byla poznána jako nějaký druh.

8 Návrhy pro rozšíření

Řešení uvedené v této práci by bylo možné do budoucna rozšířit několika způsoby, v závislosti na tom, kterým směrem by se projekt ubíral. Nabízí se např. rozšíření na více krmítek, provádění složitějších statistik nad zaznamenanými daty, trénování modelu na více obrázcích nebo vylepšit hardware, aby byla funkční hadrwarová akcelerace, což by přineslo i další možnosti k rozšíření.

Pro rozšíření na více krmítek by bylo potřeba upravit databázi a webovou aplikaci, aby podporovaly rozlišení záznamů z jednotlivých stanic a více streamů. Bylo by možné rozšířit autentizační systém, aby se přidělovala oprávnění různým lidem na různá krmítka. V nejrozšířenější podobě by bylo možné udělat projekt kolaborativní, aby dobrovolníci mohli přidat a spravovat svoje krmítko do centrální webové služby.

V této práci jsou základní statistiky určující počet výskytů druhů ptáků. Bylo by možné hledat zajímavé trendy podobné těm zmíněným v kapitole Výsledky. Šlo by také trendy propojit s informací počasí, aby se dalo zjistit, jak se liší chování ptáků v různých podmínkách.

Trénování modelu na více obrázcích by vyžadovalo z nasbíraných dat vytvořit další datasety, nebo použít otevřená data pro trénování na opravdu velkých počtech obrázků. Kód v této práci je na trénování z dalších datasetů připraven.

Vhodným rozšířením by bylo také použít jiný hardware nebo zprovoznit hardwarový akcelerátor. Dostatečně rychlá predikce by společně s lepším modelem pro detekci umožnila implementaci jiných sledovacích algoritmů, takže by pak sledování ptáků bylo přesnější.

9 Závěr

V této práci jsem vytvořil hardwarově-softwarový systém, který kamerou snímá ptačí krmítko na rodinné chalupě, rozpoznává v něm ptáky a určuje jejich druh. Statistiky a živý stream jsou zpřístupněné pomocí webové aplikace běžící na serveru na hostingu.

Pro snímání krmítka jsem použil konstrukci, na kterou jsem připevnil kameru v ochranné krabičce. Dovnitř chalupy jsem umístil počítač Raspberry Pi, který zpracovává data z kamery pomocí strojového učení a zaznamenává výsledky.

Pro účely práce jsem využil a dotvořil infrastrukturu v čele se serverem, na kterém běží webová aplikace a VPN server, sloužící k připojení Raspberry Pi k serveru a opačně.

Pro rozpoznávání jsem vytvořil složitou aplikaci, která neustále načítá data z kamery, rozpoznává je, dále pak nastaví anotace do výstupního streamu, který se konzumuje z webové aplikace. Tato aplikace také objekty sleduje a zaznamenává. Zaznamenaná data se odesílají do webové aplikace.

Pro rozpoznávání konkrétních druhů ptáků v dané oblasti jsem natrénoval vlastní model pomocí transfer learning. K tomu jsem použil nasbíraná data, která jsem původně sbíral fotopastí, po instalaci Raspberry Pi jsem přešel na sběr pomocí aplikace motionEye.

Během vytváření této práce jsem se potýkal s problémy s kamerou, jelikož jsem měl ne příliš dobře nastavený software motionEye v období sběru dat. Kamery se také často porouchávaly, musel jsem je vyměnit. Důvodem tohoto byl otvor v nepoužité horní průchodce v ochranné krabičce. Domnívám se, že tento otvor byl způsoben jedním z ptáků, kteří si na krabičku s kamerou často sedají.

Vytvořil jsem webovou aplikaci využívající framework Django s vlastním webdesignem. Jsou zde k dispozici dvě zobrazení, jedno z nich je veřejně dostupná statistika výskytu jednotlivých druhů ptáků v určitém časovém období. Dalším zobrazením je živý stream, který je chráněný přihlášením do aplikace z důvodu ochrany soukromí. Pro přímou ochranu steamovaných dat jsem propojil přihlášení poskytované frameworkem s ověřováním na úrovni HTTP serveru.

Pro tuto práci jsem zamýšlel použití hardwarového akcelerátoru pro efektivnější rozpoznávání, toho jsem však nedosáhl z důvodu nekompatibility použitého detekčního modelu. Z tohoto důvodu jsem pro sledování objektů nepoužil běžně používané sledovací algoritmy, ale místo toho jsem použil vlastní algoritmus. Přesto díky odlehčenému frameworku TensorFlow Lite byla predikce do určité míry uspokojivá.

Software vytvořený v této práci je připraven k rozšiřování a lze jej použít jako základ k podobným rozpoznávacím projektům.

Použitá literatura

[1] BLUENVIRON. *MediaMTX* [online]. c2024. [cit. 2024-05-13]. Dostupné z: https://github.com/bluenviron/mediamtx.

[2] BURHAN a Glenn JOCHER. *Export* [online]. 2023-11-12. [cit. 2024-05-14]. Dostupné z: https://docs.ultralytics.com/modes/export/.

[3] CORNELL UNIVERSITY. *Merlin Bird Id - home* [online]. c2024. [cit. 2024-05-12]. Dostupné z: https://merlin.allaboutbirds.org/.

[4] ČESKÁ SPOLEČNOST ORNITOLOGICKÁ. *Co je Ptačí hodinka* [online]. [cit. 2024-05-09]. Dostupné z: https://ptacihodinka.birdlife.cz/o-projektu/.

[5] DIXIT, Shilpa a Nitasha SONI. Comparative Study on Image Detection using Variants of CNN and YOLO. *IEEE Xplore* [online]. 2022 [cit. 2024-05-14]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8694781/.

[6] DJANGO SOFTWARE FOUNDATION. *Databases* [online]. c2005-2024. [cit. 2024-05-13]. Dostupné z: https://docs.djangoproject.com/en/5.0/ref/databases/.

[7] DJANGO SOFTWARE FOUNDATION. *Django overview* [online]. c2005--2024. [cit. 2024-05-13]. Dostupné z: https://www.djangoproject.com/start/overview/.

[8] DJANGO SOFTWARE FOUNDATION. *Internationalization and localization* [online]. c2005-2024. [cit. 2024-05-13]. Dostupné z: https://docs.djangoproject.com/en/5.0/topics/i18n/.

[9] DJANGO SOFTWARE FOUNDATION. *User authentication in Django* [online]. c2005-2024. [cit. 2024-05-13]. Dostupné z: https://docs.djangoproject.com/en/5.0/topics/auth/.

[10] FU, Yixing. *Image Classification via fine-tuning with EfficientNet* [online]. 2020-06-30. [cit. 2024-05-14]. Dostupné z: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/.

[11] GOOGLE INC. *TensorFlow Lite* [online]. [cit. 2024-05-14]. Dostupné z: https://www.tensorflow.org/lite/guide.

[12] KERAS TEAM. *About Keras 3* [online]. [2024]. [cit. 2024-05-13]. Dostupné z: https://keras.io/about/.

[13] MUNAWAR, Muhammad Rizwan, Glenn JOCHER a Ayush CHAURASIA. *Home* [online]. 2023-11-12. [cit. 2024-05-14]. Dostupné z: https://docs.ultralytics.com/.

[14] NEXT VISION LIMITED. *Picture Bird App* [online]. [cit. 2024-05-12]. Dostupné z: https://picturebirdai.com/app.

[15] PATIL, Pratik. *Object Detection using YOLOv3* [online]. 2020-05-31. [cit. 2024-05-14]. Dostupné z: https://medium.com/analytics-vidhya/object-detection-using-yolov3-d48100de2ebb.

[16] SATPATHY, Prasansha. *Deploying image classification model using the saved model in the format of tflite file and H5 file* [online]. 2021-08-07. [cit. 2024-05-14]. Dostupné z: https://prasanshasatpathy.medium.com/deploying-image-classification-model-u%20sing-the-saved-model-in-the-format-of-tflite-file-and-h5-file-92bcaf299181.

[17] SHRESTHA, Ajay a Ausif MAHMOOD. Review of Deep Learning Algorithms and Architectures. *IEEE Xplore* [online]. 2019-04-22 [cit. 2024-05-14]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8694781/.

[18] SVOBODNÝ, Hynek Václav. *Systém pro automatické rozpoznávání ptáků*. Liberec, 2023.

A Použité nástroje

Pro úpravu formulace byla použita aplikace ChatGPT verze 3.5.

B Přílohy

Kód aplikace pro rozpoznávání obrazu je k dispozici na https://github.com/hyneq/bird_identification/tree/v1.0. Součástí repozitáře je vlastní natrénovaný model klasifikace druhu ptáka.

Kód webové aplikace je k dispozici na https://github.com/hyneq/bird_identification-web/tree/v1.0.

Nasazená webová aplikace v době odevzdání a obhajoby je dostupná na adrese https://birds.hyneq.cz/