FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Detekce vybrané klíčové činnosti při pozemním odbavení letu na základě záznamů z bezpečnostních kamer
Student:	Teodor Beneš
Vedoucí:	Ing. Josef Pavlíček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Umělá inteligence 2021
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Pozemní odbavení letu, které probíhá na letištní stojánce, zahrnuje ze strany odbavovací společnosti (handling) mnoho fyzických úkonů, které jsou klíčové pro rychlý a bezpečný průlet na letišti. V současné době není z pohledu letiště celý proces odbavení dostatečně digitalizován, tj. neexistuje evidence o některých prováděných činnostech při odbavení letu. Zavést takovou evidenci manuálně by však zároveň pro pracovníky letiště představovalo neúměrnou zátěž. Cílem této práce je tedy automatizovaná identifikace těchto činností na základě rozpoznávání obrazu a detekce objektů ze záznamů bezpečnostních kamer.

Postupujte dle následující metodiky:
1)Seznámení se s původním prototypem řešení detekce objektů z bakalářské práce BLAŠKO, Oliver. Detekce klíčových objektů na letištní stojánce z bezpečnostních kamer 2021.
2) Proveďte rešerši současných podobných řešení, zaměřte se na slabé stránky studovaného prototypu, definujte je a pokuste se nalézt buď existující řešení nebo alespoň metodiku, jak problém řešit.
3)Seznamte se se zdrojovými daty (dodá vedoucí specialista z firmy Profinit Praha).
4) Na základě výsledků rešerše a analýzy dat navrhněte budoucí řešení z oblasti detekce událostí:

4. a) Příjezd letadla na stojánku,
4. b) Pokuste se na základě zdrojových dat detekovat uzavření kabiny letadla (ze

---

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 5. února 2024 v Praze.

strany, ze které je snímáno kamerou) a navrhnout, jak z dat rozpoznat další objekty typu připojení letadla kabelem, zákolník a další, vyplývající z rešerše dat.
5) Vytvořte prototyp. Zdrojové kódy prototypu budou uloženy v datovém repozitáři typu GitHab atd.
6) Otestujte prototyp na základě bodu 4.a. Popište úspěšnost systému v rozpoznání (versus halucinace) události.
7) Definujte závěry a proveďte diskusi v oblasti úspěšnosti Vašeho řešení v porovnání s jinými.

Bakalářská práce

DETEKCE VYBRANÉ KLÍČOVÉ ČINNOSTI PŘI POZEMNÍM ODBAVENÍ LETU NA ZÁKLADĚ ZÁZNAMŮ Z BEZPEČNOSTNÍCH KAMER

Teodor Beneš

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Josef Pavlíček, Ph.D.
16. května 2024

Odkaz na tuto práci: Beneš Teodor. *Detekce vybrané klíčové činnosti při pozemním odbavení letu na základě záznamů z bezpečnostních kamer*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu Ing. Josefu Pavlíčkovi, Ph.D. za vedení bakalářské práce, oponentovi Mgr. Janu Paláškovi, který mi dával dobré rady ohledně technické části práce a Ing. Petru Filasovi, který přispíval svými znalostmi o fungování letiště.
Také bych chtěl poděkovat fakultě informačních technologií ČVUT za všechny znalosti, které jsem díky ní získal a za zázemí, které mi umožnilo se studiu věnovat naplno.
V neposlední řadě patří mé díky rodině, přítelkyni a přátelům za jejich podporu a trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16. května 2024

Abstrakt

Tato bakalářská práce se zabývá detekcí klíčových událostí na letišti. Těmito událostmi jsou zaparkování letadla na vyznačeném místě, připojení tunelu pro průchod pasažérů do letadla a otevření dveří zavazadlového prostoru. Využit byl model YOLO, který úspěšně zaznamenává zadané klíčové události, s výjimkou problémů, způsobených nepříznivým počasím. Například déšť může pokrýt objektiv kamery a zapříčinit zkreslení záznamu. Za běžných podmínek model detekuje průběh činností s průměrnou přesností ±15 s.

Klíčová slova detekce objektů, letištní stojánka, strojové vidění, YOLO, labeling, konvoluční neuronové sítě

Abstract

In this bachelors thesis we are going to look into detection of key actions on the airport apron. We are going to focus on three main situations. Detecting when an airplane parked, when it opened its cargo doors and when the jet bridge connected to the airplane. I'm going to use the YOLO object detection model to detect those key situations and build more logic on top of it. The model was very much successful. The only problem arises when camera lens gets covered with rain. Then the image is blurry and model can't work properly. Aside from that we got a ±15 s accuracy on average when detecting those key situations.

Keywords object detection, airport apron, computer vision, YOLO, labeling, convolutional neural networks

Seznam zkratek

AI	Artificial Intelligence - Umělá Inteligence
CNN	Convolutional Neural Network - Konvoluční Neuronová Síť
YOLO	You Only Look Once - Model na bázi jednokrokové detekce
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
BoF	Bag of Freebies
BoS	Bag of Specials
E-ELAN	Extended Efficient Layer Aggregation Network
IoU	Intersection over Union
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Kapitola 1
Úvod

V této kapitole bych chtěl zdůvodnit svou motivaci pro výběr tohoto tématu. Přehledně sepíšu cíle bakalářské práce a nakonec přidám přehled pojmů užívaných v této práci.

1.1 Motivace

Obor umělé inteligence zaznamenal v posledních letech nebývalý vzestup. Máme možnost ji využívat k zefektivňování práce v mnoha oblastech lidské činnosti, což se projevuje i v každodenním životě lidí. Svět je čím dál tím rychlejší, jednotlivec nemá šanci obsáhnout a pochopit drahocenné informace, které ho bombardují ze všech stran. Může to být ubíjející, ale přesto se všechny tyto informace dají zpracovat do uchopitelnější formy, ve které mohou přinášet užitek všem.

Umělá inteligence nám ulehčuje předpověď počasí, říká nám, jak by se podle známých faktů mohla vyvíjet burza a filtruje nám obsah mediálních zdrojů na ty, které by mohly zajímat právě nás. S trochou nadsázky skoro každému z nás, ať si to uvědomujeme či ne, ulehčuje umělá inteligence život.

Mnoho z nás alespoň jednou za život letělo letadlem. A nemalé procento z těchto lidí jistě zažilo i vícehodinové čekání na zpožděné letadlo, do kterého právě obsluha letiště nakládala zavazadla, tankovala a zajišťovala, aby dané letadlo bylo plně připravené k odletu. Každá prodleva při odbavování letadla má za následek prodlužování zpoždění. A takto nakumulované zpoždění stojí letiště obrovské množství peněz. Každá minuta užívání jedné letištní stojánky má hodnotu kolem sta dolarů [1]. Proto je snaha situace způsobující prostoje minimalizovat. Pro člověka je komplikované ohlídat vyšší desítky stojánek a zajistit tak, aby odbavování všude probíhalo hladce. Proto je snaha kontrolu letištní stojánky automatizovat. K tomu můžeme využít data z kamer umístěných na letišti.

Se zpracováním těchto dat nám opět pomůže umělá inteligence. Využívat budeme techniku zpracování obrazu zvanou object detection. Jak již název napovídá, umožňuje nám detekci objektů na obrázcích či videích. Nejvíce jsou pro tento účel využívány modely na bázi konvolučních neuronových sítí, které zažívají boom v posledních deseti letech [2] a velmi rychle se posouvají kupředu. Každým rokem jsou vyvíjeny nové modely, které jsou vždy o krok napřed před těmi loňskými. Vylepšení se netýká pouze výsledků, které jsou modely schopné nám předat, ale také umožňují rychlejší a jednodušší trénování za použití nižší výpočetní kapacity.

Model, který v této bakalářské práci natrénuji, mi umožní detekovat klíčové objekty. Ze vzájemných vztahů mezi těmito objekty jsem pak schopen určit, k jakým situacím v danou chvíli na stojánce dochází. Důraz bude kladen na jednoduchost a transparentnost interpretace výsledků. Lidé mají velkou nedůvěru k novým věcem, když neví, jak fungují. Jakmile ale pochopí, co mohou očekávat a jak celá věc funguje, rádi využívají benefity, které jim přináší.

1.2 Cíle

Cílem bakalářské práce je detekce následujících činností:

1. zaparkování letadla na vyznačeném místě
2. otevření dveří zavazadlového prostoru
3. připojení nástupního mostu k letadlu

Cílit budu na real timové zpracování těchto činností, a to s co nejpřesnějším ohraničením jejich začátku a konce. Také je podstatné snížit na minimum výskyt hrubých chyb v detekci – detekování něčeho, co se v datech vůbec neděje.

1.3 Přehled pojmů

- letištní stojánka – místo kde probíhá příprava letadla na odlet
- tunel pro nástup pasažérů, nástupní most – propojení nástupní haly s letadlem
- YOLO – zkratka z „you only look once“ – model fungující na bázi jednofázové detekce
- anotace, labelling – označování objektů/událostí ve videích, z těchto informací se pak model učí
- konfidence – jistota, s jakou model provádí detekci

1.4 Popis kapitol

- **Rešerše** – rozbor řešení týkajících se podobné tématiky, volba nástrojů, přehled fungování modelu YOLO a popis shlukovacího algoritmu DBSCAN
- **Analýza** – přiblížení datasetu, analýza možností postupu, metody pro zlepšování robustnosti výsledků a zadefinování metrik
- **Implementace** – zpracování dat, trénování modelu, implementace metrik a ladění řešení
- **Testování** – otestování finálního řešení na testovacích datech, diskuse výsledků, popis silných a slabých stránek
- **Shrnutí poznatků** – přehledné shrnutí nejdůležitějších informací získaných z bakalářské práce
- **Závěr** – zhodnocení výsledků bakalářské práce

Kapitola 2
Rešerše

V této kapitole rozeberu bakalářskou práci Bc. Olivera Blaška, na kterou budu v této práci navazovat. Zmíním zajímavé cesty, jak řešit problematiku detekce na letišti. Proberu zde komerční řešení využívaná na mnoha letištích po celém světě. Následně přejdu k popisu nástrojů využívaných v této bakalářské práci. Zmiňovat budu nástroj na labelling dat, výběr modelu, udělám krátký přehled jeho hyperparametrů a zmíním možnosti, jak tyto hyperparametry ladit. Nakonec popíšu shlukovací algoritmus DBSCAN využívaný pro analýzu pozic kola letadla.

2.1 Návaznost na bakalářskou práci Bc. Olivera Blaška

V této práci navazuji na bakalářskou práci Bc. Olivera Blaška z roku 2021. Autor se zde věnoval detekování všech objektů viditelných ze zdrojových dat. Detekce zahrnovala následující objekty:

- letadlo
- dveře zavazadlového prostoru
- tunel pro průchod osob
- vozidla s palivem a elektřinou
- auta vozící boxy s nákladem a i samotné boxy
- pushback – vozidlo, které odtlačuje letadlo ze stojánky

Díky této práci víme, že neuronové sítě umí bez problému detekovat objekty na letištní stojánce. Autor v práci využíval YOLOv4.[3]

2.2 Rešerše současných řešení

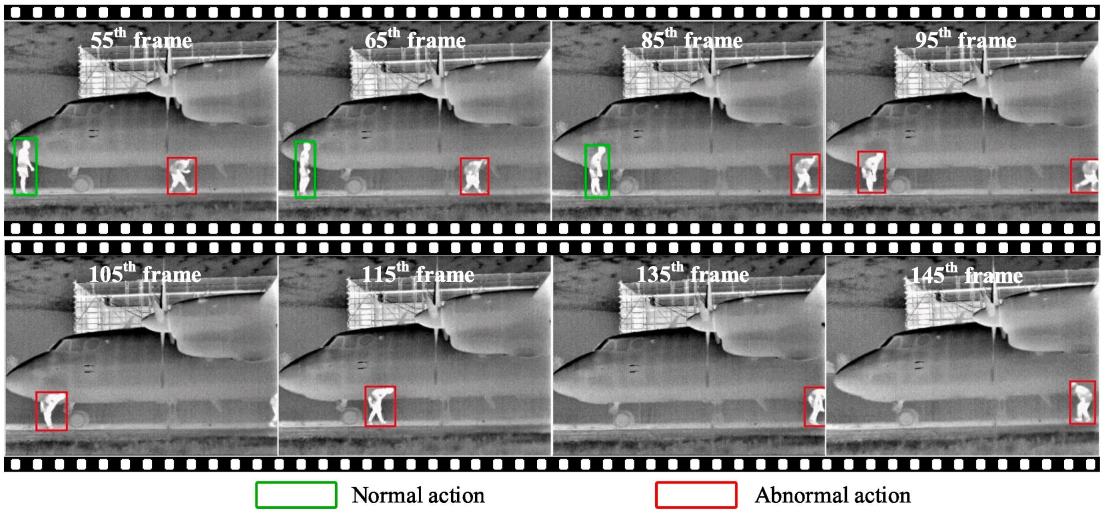
V této sekci proberu řešení zabývající se podobnou tématikou jako tato práce. Zmíním jejich výhody a nevýhody a uvedu, jak je možné se z nich inspirovat.

2.2.1 Výzkumné práce

V této subsekci se zaměřím na možnosti řešení prezentované jako součást výzkumu, nebo jako součást vzdělávacích programů.

2.2.1.1 Detekce činností lidí na bázi dat z termální kamery

Zajímavým řešením z podobné oblasti bylo využití infračervených kamer pro detekování činností osob na letišti. Snaha je rozlišit běžné činnosti od činností podezřelých, jako je například skrývání se pod letadlem. Snímky prezentované v tomto článku můžeme vidět na obrázku 2.1. Naše řešení se nebude týkat osob, ale myšlenky prezentované v tomto článku by se daly využít pro jeho zlepšení. Infračervené kamery mají výhodu, že nejsou tak závislé na počasí a denní době. To znamená, že jsou schopné detekovat objekty i v husté mlze nebo v naprosté tmě. Jejich velkou nevýhodou ale je, že na rozdíl od běžných kamer, nejsou běžně instalovány na letištích. Z toho plyne, že chybí datasety pro trénování a aplikování řešení může být komplikovanější. Každé letiště, které by mělo zájem o využívání takového řešení by muselo zainvestovat do pořízení infračervených kamerových systémů.[4]

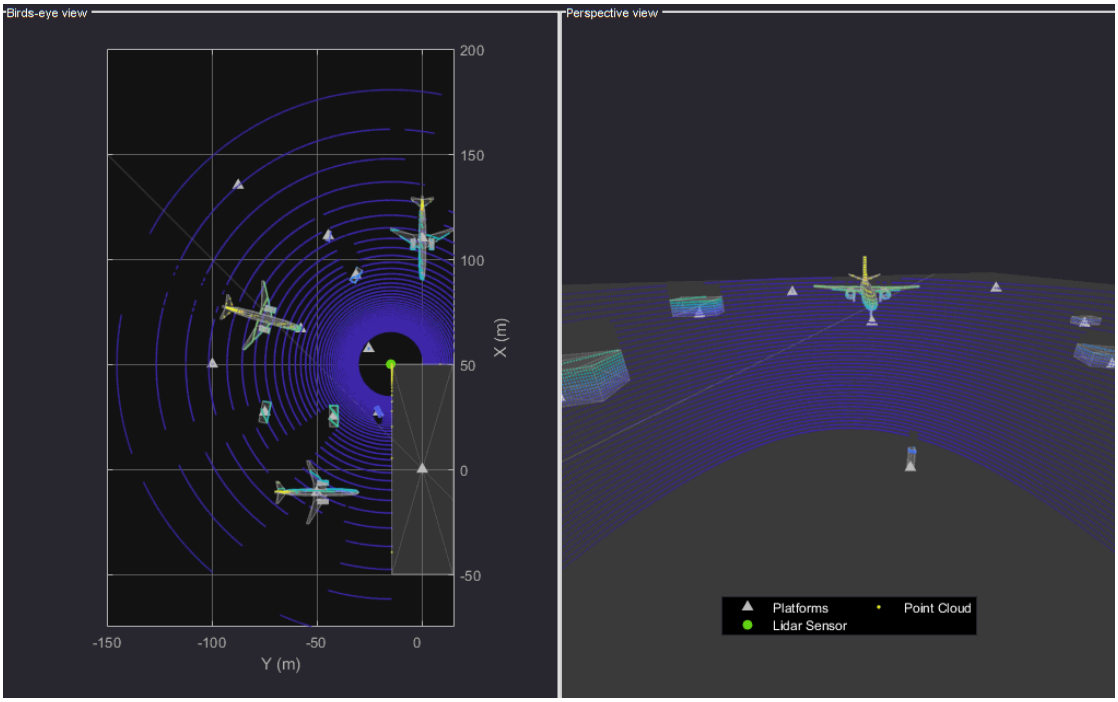


Obrázek 2.1 Detekce z termálních kamer. Zdroj: [4]

2.2.1.2 Využití lidaru na letištní stojánce

Další technologií, která není tak závislá na dobrých podmínkách a viditelnosti je lidar. Jedná se o způsob měření vzdálenosti pomocí laseru. V článku [5] můžeme vidět ukázku simulace využití lidaru v matlabu. Informace takto nasimulované jsou používané k demonstraci monitoringu objektů, pohybujících se na letištní stojánce. Na obrázku 2.2 můžeme vidět ukázku dané simulace.

Lidar může být vhodným kandidátem pro sběr informací o tvarech a polohách objektů na letištní stojánce. Nevýhodu vidím opět v nutnosti pořizování dalšího vybavení pro každou letištní stojánku. Ovšem pokud by se takováto investice na letišti učinila, byla by možnost využívat lidarové informace k doplnění informací z kamer. To by dobře ovlivnilo spolehlivost řešení převážně v situacích, kdy je vidění běžných kamer zkresleno, nebo znemožněno vnějšími faktory jako je déšť, tma a podobné.



Obrázek 2.2 Ukázka simulace lidaru na letištní stojánce. Zdroj: [5]

2.2.2 Komerční řešení

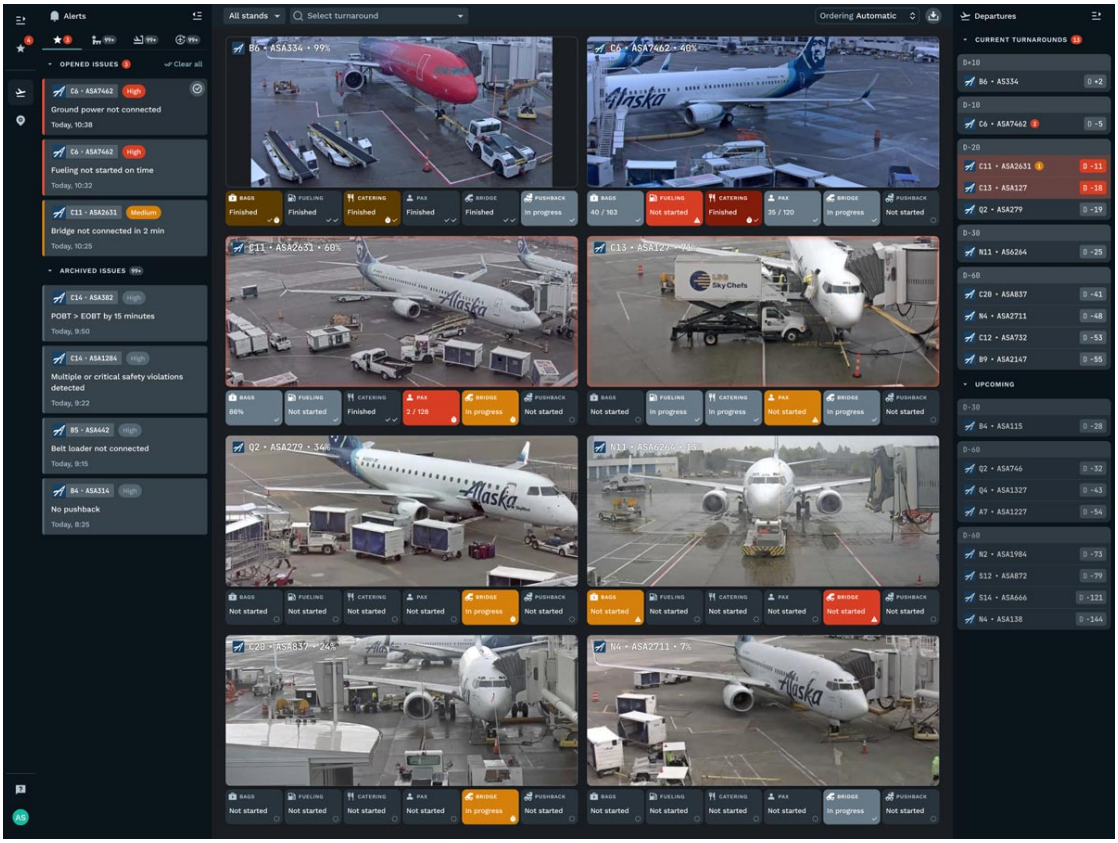
Mnoho letišť po celém světě využívá nástroje, které vytvořily firmy zmiňované v této subsekci. Díky efektivnímu zpracování informací ušetří letiště mnoho času i peněz. V této sekci se inspiruji postupy, které by se daly využít v našem řešení a zhodnotím jejich výhody a nevýhody.

2.2.2.1 Assaia

Nejpodobnější řešení problematiky této bakalářské práce nabízí firma Assaia. Jedná se o komerční produkt, u kterého nejsou zveřejňovány podrobnosti ohledně jeho fungování na pozadí. Pouze víme, jaké zhruba funkce má a jak bychom k řešení problému mohli přistupovat. Z jejich webových stránek se mi podařilo pochopit následující informace. Firma Assaia ve svém řešení detekuje přímo stav, kdy je nástupní most připojen a kdy jsou otevřené dveře zavazadlového prostoru. Z prezentace jejich výsledků není vidět, jestli detekují samotný stav zaparkování letadla. Oproti našemu řešení navíc detekují nakládání a vykládání zavazadel, doplňování potravin, připravenost vozidla na odtlačení letadla ze stojánky a další. Na obrázku 2.3 je náhled uživatelského rozhraní řešení firmy Assaia.[6]

Hlavní nevýhodou řešení firmy Assaia vidím v ceně na zpracování projektu. Cenu řešení firmy Assaia odhaduji z výňatku „*From work that we have been doing with one of the major airports in the US we have recorded an average 4 minute reduction time in GPU/ACU connection time. This saves airlines more than $7 and more than 5 kg of CO2 emissions per flight. For a medium to large airport this equates to $1-$1.4 million per year which is a multiple of the cost of Assaia's TurnaroundControl!*“ [7].

Píší, že střední letiště ušetří $1-$1.4 milionu ročně a že tato částka je násobně vyšší než ta, kterou si účtují. Může to být tedy okolo sta tisíc dolarů. V této bakalářské práci se tedy pokusím zpracovat jednodušší, méně komplexní řešení, které je rychlejší na vytvoření a nasazení.

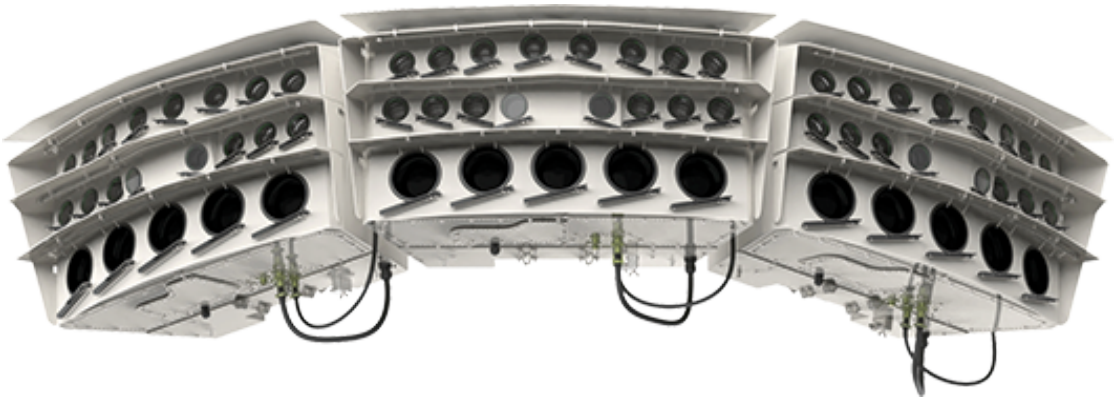


Obrázek 2.3 Ukázka fungování řešení od firmy Assaia. Zdroj: [6]

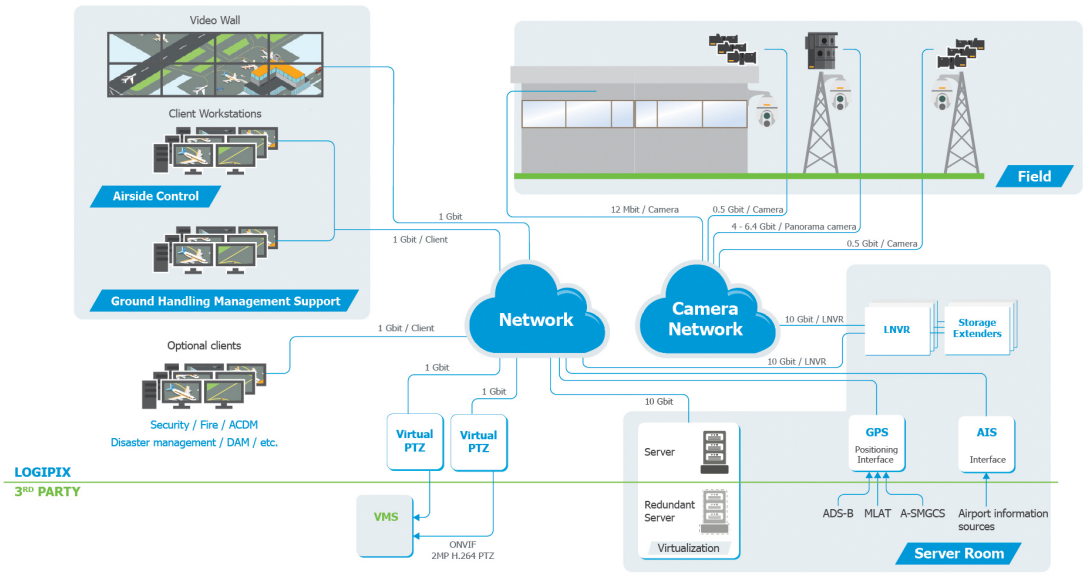
2.2.2.2 Logipix

Firma Logipix nabízí množství produktů využívajících object detection. Tři z nich se týkají oblasti letiště a jedno z nich je přímo detekování objektů na stojánce pro zrychlení odbavení letadla. Tato firma klade velký důraz na kvalitu dat, která jejich řešení využívají. Na jejich webových stránkách je vidět množství pokročilých zařízení pro získávání obrazu, jako jsou například panoramatické a dálkové kamery schopné zaznamenávat denní i noční podmínky ve viditelném i infračerveném světelném spektru. Tyto kamery jsou také vybaveny čistícími systémy. Umí se samy odmrazovat a čistit své objektivy od deště a nečistot pomocí stěračů a ostřikovačů. Tyto kamery můžeme vidět na obrázku 2.4. Na obrázku 2.5 také můžeme vidět diagram fungování jejich řešení na pozadí.[8]

Stejně jako u firmy Assaia, nevýhoda řešení firmy Logipix spočívá ve velkých nákladech na zpracování a k tomu se ještě přidávají výdaje na pořízení speciálních kamer a dalšího hardwaru.



Obrázek 2.4 Ukázka jedné z kamer logipix. Zdroj: [8]



Obrázek 2.5 Ukázka fungování řešení firmy logipix. Zdroj: [8]

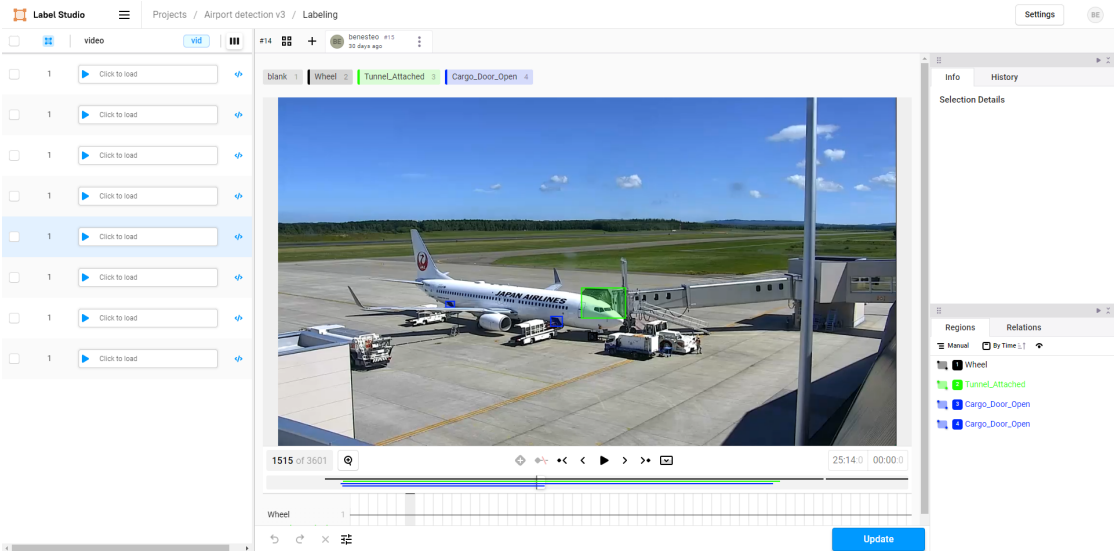
2.3 Volba nástrojů

Prvním krokem k vypracování dobrého řešení je výběr ideálních nástrojů pro daný účel. V této sekci odůvodním výběr nástroje pro labelling dat a výběr ideálního modelu pro naše použití.

2.3.1 Nástroj pro labelling

Vzhledem k velkému množství dat v datasetu je potřeba efektivní způsob, jak tato videa oanotovat. Jako nejpříhodnější mi přišel open-source nástroj Label Studio [9]. Umožňuje anotovat různé druhy dat, včetně anotace videí pro object detection. Velkou výhodou tohoto nástroje je využití lineární interpolace pohybů bounding boxů mezi jednotlivými keyframy. Pokud se tedy letadlo pohybuje například dvacet sekund stejnou rychlostí a stejným směrem, stačí pouze ozna-

čit bounding boxy na začátku a na konci tohoto pohybu a pozice bounding boxů v pohybu se dopočítá. Uživatelské rozhraní label studia můžeme vidět na obrázku 2.6.



Obrázek 2.6 Ukázka user interface label studia. Zdroj: vlastní screenshot

2.3.2 Volba modelu

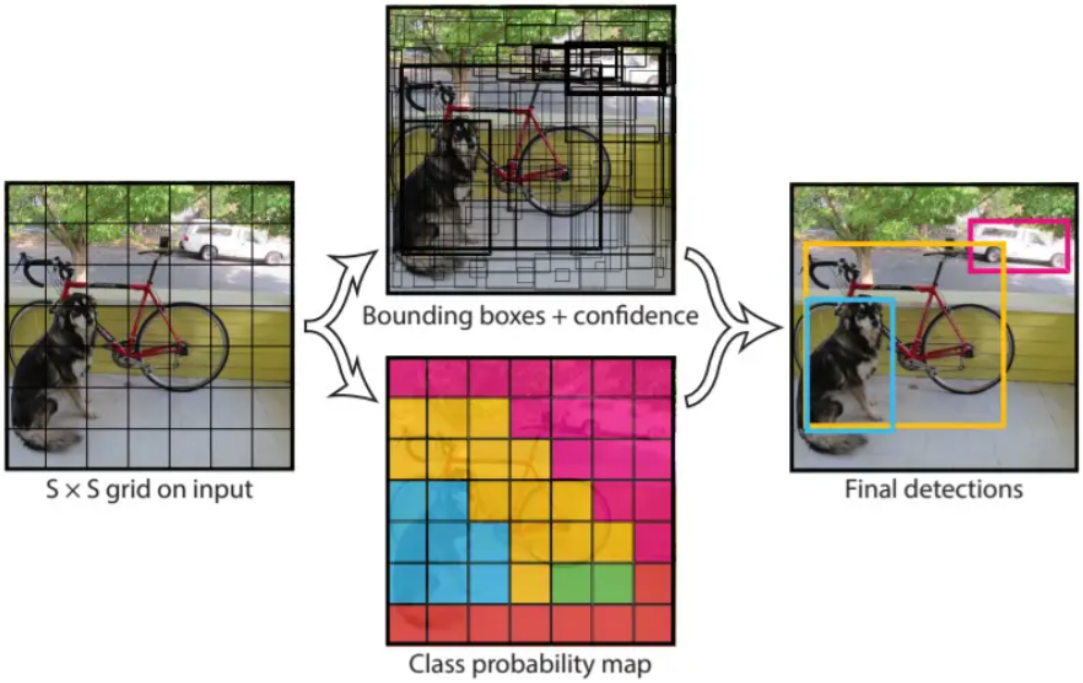
Model užívaný k detekci objektů je nejdůležitější součástí našeho řešení. Je tedy nutné vybrat správný model pro naše účely. V této sekci zdůvodním, proč jsou pro nás výhodnější modely na bázi jednofázové detekce a následně zvolím ideální model pro real timové použití.

2.3.2.1 Jednofázová vs Dvoufázová detekce

Vzhledem k tomu že bychom chtěli tento systém používat real-time, nebo near real-time, tak jsou příhodnější modely využívající jednofázovou detekci. Ta totiž probíhá výrazně rychleji. Její nevýhoda spočívá v horší přesnosti, ale rozdíl není pro nás zásadní. Hlavními představiteli tohoto přístupu jsou YOLO, efficientdet a RetinaNet. Jako nejlepší pro real-time aplikace je bráno YOLO, jak můžeme vidět na obrázku 2.7. YOLO bude tedy primární model využívaný v této bakalářské práci. Obrázek 2.8 ukazuje obecné fungování modelů YOLO.[10]

Model	Type	Pros	Cons
YOLO	One-stage	• Very fast processing speed • Simple architecture • Good for real-time applications	• Can be less accurate than two-stage detectors • May struggle with small objects
EfficientDet	One-stage	• Excellent balance of accuracy and speed • Efficient backbone network • Good for mobile and resource-constrained environments	• May still lag slightly behind top two-stage detectors in accuracy • Can struggle with very small objects
RetinaNet	One-stage	• High accuracy • Addresses class imbalance well	• Can be slower than YOLO • More complex architecture
Faster R-CNN	Two-stage	• High accuracy • Good for detecting small objects	• Slower than one-stage detectors • More complex training process
Mask R-CNN	Two-stage	• High accuracy for instance segmentation • Provides pixel-level masks for objects	• Even slower than Faster R-CNN • More computationally expensive

Obrázek 2.7 Porovnání modelů. Zdroj: [10]



Obrázek 2.8 Fungování modelů YOLO. Zdroj: [11]

2.3.2.2 Které YOLO zvolit?

Modely YOLO se velmi rychle vyvíjí. V této sekci ve zkratce popíšu vylepšení, která jednotlivé verze YOLA přinesla a zdůvodním výběr modelu YOLOv8.

V době psaní této práce je nejnovějším zaběhlým modelem YOLOv8. YOLOv9 bylo čerstvě vydáno, ale prozatím nejsou k disposici všechny velikosti modelu. Budu tedy pracovat s YOLOv8. Verze 8 je vylepšením oproti předchozím verzím. Dosahuje lepších výsledků s menším počtem parametrů. Méně parametrů, zjednodušeně řečeno, znamená rychlejší trénování a inferenci a nižší nároky na paměť. Vzhledem k tomu, že je předpoklad, že výsledný model bude využíván na lev-

nějších zařízeních s nižším výkonem, využiji nejmenší verzi a to YOLOv8n. N na konci znamená nano.[12]

YOLOv1 bylo revoluční svým přístupem k detekci objektů. Nahlíželo na problém jako na regresní úlohu, sestrojilo mapu pravděpodobností výskytu objektů v obrázku a pomocí té následně generovalo bounding boxy.

YOLOv2 bylo schopné detekovat přes 9 000 kategorií objektů. V2 také přinesla takzvané "anchor boxy"– předdefinované bounding boxy, které model využíval ke zpřesnění detekce pozice objektů.

YOLOv3 využívalo logistické klasifikátory namísto softmaxu a binární cross entropie. To přineslo další zrychlení a zpřesnění modelu.

YOLOv4 bylo vydáno v roce 2020 Alexeyem Bochkovskiym a jeho spolupracovníky. Přínosem YOLOv4 bylo zavedení takzvaných „Bag of Freebies“ (BoF) a „Bag of Specials“ (BoS). BoF obsahoval množství technik augmentace dat. Těmi byly CutMix, CutOut, Mixup a Mozaika. Mozaika vzala 4 různé obrázky z trénovací množiny a poskládala je dohromady. BoS zahrnuje nelineární aktivační funkce a „skip connections“.[13]

YOLOv5 byl první model od Ultralytics. Implementován v PyTorch. Novinkou bylo využití „Cross-stage Partial Connection“ bloků. Ty vylepšují propagaci gradientů a snižují výpočetní náročnost. V5 také znamenala přechod of .cfg na .yaml soubory pro konfiguraci modelu.[14]

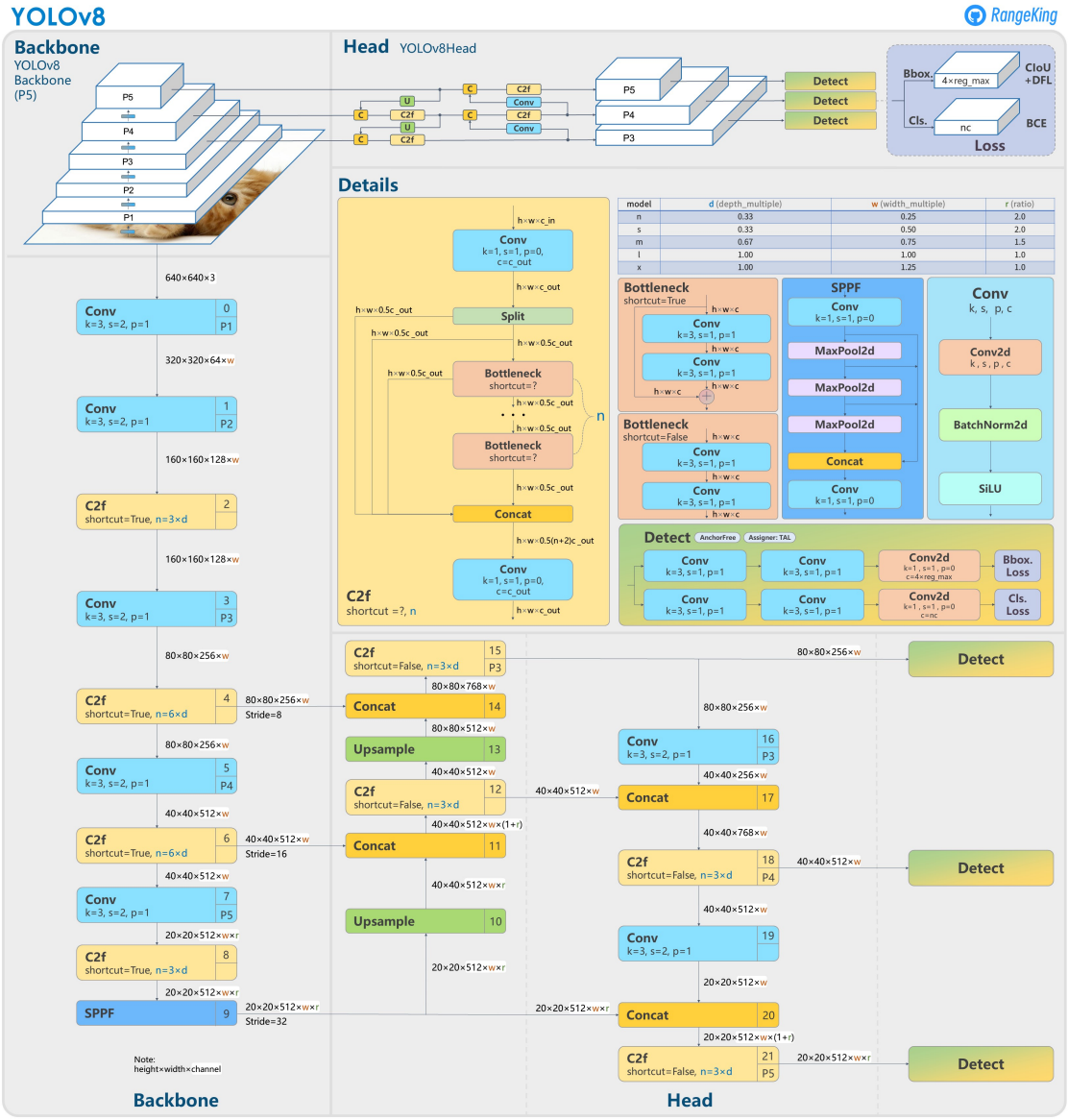
YOLOv6 funguje se dvěma hlavami – „decoupled head“. Jedna se stará o správné umístění bounding boxů a druhá o klasifikaci objektů uvnitř.

YOLOv7 využívá „Extended Efficient Layer Aggregation Network (E-ELAN)“. Umožňuje modelu naučit se rozličné znaky, což zlepšuje trénování modelu. Model také umožňuje měnit rychlost inference díky metodě „compound scaling for concatenation-based models“.

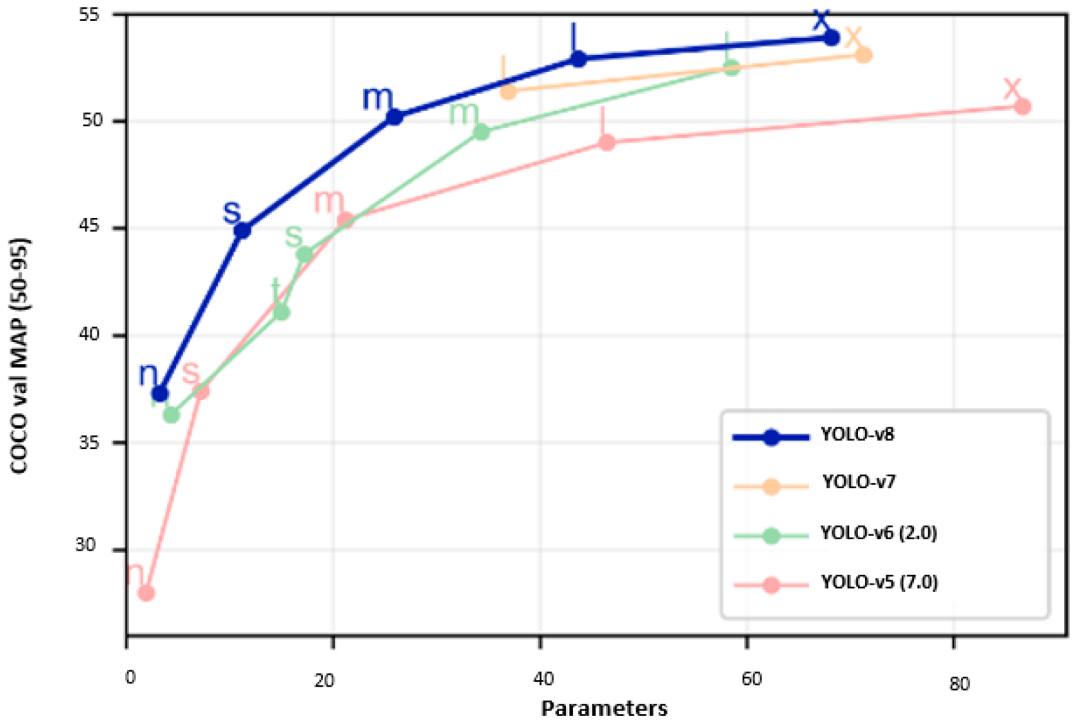
YOLOv8 je opět výtvorem Ultralytics. Jeho používání je velmi jednoduché díky python balíčku od ultralytics a také díky možnosti využívání modelu přímo z příkazové řádky. YOLOv8 je schopné provádět klasifikaci, detekci a segmentaci obrázků. Strukturu YOLOv8 můžeme vidět na obrázku 2.9.

- Stejně jako v4, v8 využívá tvorbu mozaiky pro lepší zobecnění trénovacích dat.
- Z v6 přebírá decoupled head a fungování na bázi „Anchor-Free“ detekce.

YOLOv9 je nejnovější verzí k disposici. Přináší „Programmable Gradient Information (PGI)“, které výrazně zlepšují fungování modelu i při nižším počtu parametrů. Aktuálně jsou k disposici pouze 2 největší verze YOLOv9.[15]



Obrázek 2.9 Struktura modelu YOLOv8. Zdroj: [16]



Obrázek 2.10 Porovnání modelů. Zdroj: [12]

2.4 Hyperparametry trénování modelu

Hyperparametry trénování modelu mají velký vliv na rychlost trénování i výsledné schopnosti natrénovaného modelu. V této sekci popíšu nejzásadnější hyperparametry používané pro úpravu vlastností trénování modelu.

2.4.1 Epochs a patience

Parametr **epochs** nastavuje počet trénovacích epoch. Toto číslo můžeme nastavit vysoké a jakožto zastavovací kritérium použít následující parametr **patience**. Ten nastavuje počet epoch, po kterých zastavíme trénování, pokud se nezlepšuje chování modelu na validačních datech. Pokud se například model po 50 epochách začne přeučovat (model se zlepšuje na trénovací množině, ale na validační už jeho metriky klesají), zastavíme trénování a ponecháme si model s nejlepším skóre na validační množině. Tento parametr je nutné nastavit dostatečně vysoký, aby nedošlo zbytečně k předčasnému zastavení. Pokud bychom nastavili patience kupříkladu na 5, může se stát, že model bude 5 epoch stagnovat, trénování by bylo zastaveno a nebylo by modelu dovoleno trénovat dále, i když by měl kapacitu se dále zlepšovat. Chceme si tedy být jistí, že k dalšímu zlepšení nedojde. V našem případě může být rozumnou volbou 20 epoch. Podle náročnosti trénování a velikosti modelu se může i 50 epoch ukázat jako rozumná hodnota.

2.4.2 Close mosaic

Jak jsem již zmiňoval v sekci 2.3.2.2, model YOLOv8 využívá mozaiky při trénování. Ty pomáhají modelu při generalizaci. Model skládá obrázky z trénovací množiny do mozaiky a na té se pokouší

provádět detekci. Parametr **close_mosaic** udává počet posledních epoch před koncem trénování, ve kterých už mozaiku nemá používat. Posledních pár epoch ustálí model neaugmentovanými daty.

2.4.3 Normalizační metody

2.4.3.1 Batch size

Při používání GPU pro trénování jsme závislí na interní paměti dané grafické karty. Tento parametr nám umožňuje nastavit batch size podle velikosti dostupné paměti. Pokud nám dojde paměť karty, dojde k přerušení trénování. Proto má náš model možnost tento parametr nastavit dynamicky pomocí **batch=-1**. Model pak batch size upravuje podle aktuální volnosti paměti na GPU. Po zpracování jedné dávky model přenastavuje své vnitřní parametry podle chyby, kterou udělal. Tento parametr spolupracuje s parametrem nbs na vyrovnání batch normalizace.

2.4.3.2 Nominal batch size

Parametr **nbs** neboli nominal batch size simuluje větší batch size pomocí násobení ztráty koeficientem rovným poměru nominal batch size a batch size. Tak dosahujeme efektu, že i menší batch ovlivňuje model tak, jak by ho ovlivňovala větší batch.

Nastavováním tohoto parametru upravujeme takzvanou batch normalizaci. Ta je výhodná, protože nechceme, aby se model měnil po každém obrázku, ale chceme, aby se naučil poznávat souvislosti mezi více obrázky.

2.4.3.3 Dropout

Parametr **dropout** ovlivňuje míru náhodného vynechání spojů v neuronové síti. Vynechávání spojů zamezuje modelu, aby se stal závislým na určitých znacích objektu. U našeho modelu by se například mohl naučit, že letadla bývají jednobarevná. Při příjezdu letadla s barevnými nápisy by se mohlo stát, že nebude správně klasifikováno. Dropout při trénování občas takovou podstatnou informaci vynechá, a tak si model musí poradit i pomocí dalších faktorů, jako je například obrys a podobné.

2.4.4 Velikost snímků pro trénování

Větší velikost snímků pomáhá při detekci menších objektů ve videu. Upravujeme ji parametrem **imgsz** a to jak při trénování, tak při predikci. V našem případě i menší rozlišení je schopno detekovat všechny objekty, které nás zajímají. Jediný problém při nižším rozlišení činní detekce předního kola, když se letadlo nachází ve větší vzdálenosti od stojánky. Nižší rozlišení má ovšem za výhodu výrazně rychlejší trénování a inferenci. Velmi tak záleží na důležitosti detekce vzdálenějšího kola a potřebnosti rychlejší detekce při užití výpočtu detekce na levnějších zařízeních. YOLO umožňuje trénování a detekci na obrázcích, jejichž velikosti v pixelech jsou násobky 32. YOLO si obrázky automaticky resizuje podle potřeby.

2.4.5 Freeze

Parametr **freeze** nám umožňuje „zmrazit“ předtrénované vrstvy a dotrénovat pouze posledních pár vrstev na naše konkrétní použití. Zmražené vrstvy nijak nemění své parametry. Nestane se nám, že bychom pokazili vlastnosti, které se model naučil v předchozích fázích trénování. Také naše trénování je díky tomu rychlejší, protože upravujeme nižší počet vrstev.

2.4.6 Nastavení koeficientů chyb

YOLOv8 má 2 parametry upravující koeficienty různých druhů chyb. Parametr **cls** upravuje koeficient klasifikace detekce bounding boxu a parametr **box** upravuje koeficient chyby popisující přesnost lokalizace bounding boxu. Vzájemnou úpravou těchto koeficientů pak můžeme modelu dát najevo, že se má soustředit spíše na přesnost pozic objektů, nebo přesnost klasifikace daných objektů.

2.4.7 Optimizery a rychlosti trénování

2.4.7.1 Optimizer

Optimizer upravuje způsob učení modelu. Při nastavení **optimizer='auto'** si náš model optimizer vybírá podle velikosti datasetu a podle zvoleného počtu epoch a velikosti datasetu. Při vyšším počtu dat a epoch volí SGD a při nižším AdamW. Článek [17] porovnává chování optimizerů na datasetu seattle weather data set.

2.4.7.2 Learning rate

Parametr learning rate určuje velikost kroků dělaných v gradientním sestupu. Pokud jsou tyto kroky moc velké, dochází k oscilacím v metrikách modelu. Pokud je learning rate nastavený jako moc malý, trénování je zbytečně pomalé a může dojít k zaseknutí. Yolo umožňuje nastavení parametrů **lr0** a **lrf**. Parametr **lr0** nastavuje počáteční learning rate a **lrf** nastavuje koeficient cílového learning rate. Cílový learning rate je pak vypočítán jako násobek **lr0** a **lrf**.

2.4.7.3 Momentum

Momentum neboli hybnost umožňuje rychlejší učení neuronových sítí. Jedná se o simulaci pohybu směrem k optimálnímu řešení. Gradient násobený koeficient přidáváme ke gradientům z přechozích kroků a tím je akumulujeme. Parametr **momentum** upravuje koeficient akumulace hybnosti.

2.5 Ladění hyperparametrů

Pro natrénování nejlepšího možného modelu je nutné najít správnou kombinaci hyperparametrů, která zajistí, že se model bude chovat přesně podle našich představ. Problémem ladění hyperparametrů je, že kombinací je nekonečně mnoho a není možné vyzkoušet všechny. Existují ale nástroje, které se pokouší o zjednodušení tohoto problému. V této sekci popíšu ladění hyperparametrů a zmíním nástroje využívané pro ladění hyperparametrů u YOLOv8.

Ladění hyperparametrů je výpočetně velmi náročné. Je třeba vyzkoušet různé kombinace a pro každou kombinaci natrénovat model. Bohužel neexistuje univerzální nejlepší kombinace hyperparametrů. Jejich volba závisí na tom, jaký model trénujeme, jak rozmanitý je dataset a také na jakých datech následně chceme model nasazovat. Pokud je trénovací dataset málo rozmanitý, ale chceme, aby se model na nových datech choval co nejlépe, je dobré zvolit větší míru regularizace, i když to nemusí být nejlepší na našich trénovacích datech.

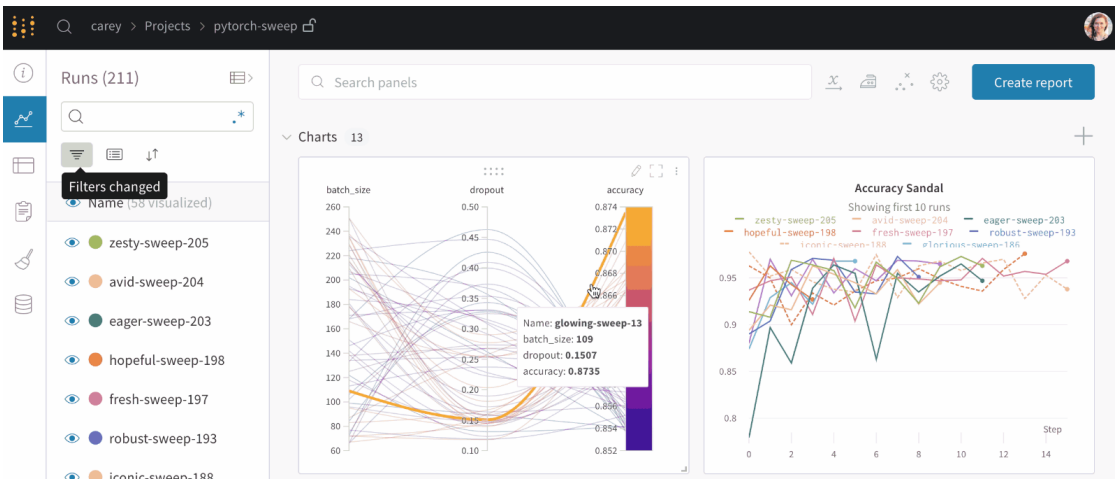
Ultralytics umožňuje ladit hyperparametry modelu pomocí genetických algoritmů. Spuštění takového ladění můžeme vidět na ukázce 2.1 [18]. Ultralytics také umožňuje využití knihoven jako **ray tune** a **weights & biases**. **Ray tune** můžeme jednoduše použít nastavením parametru **use_ray=True**, jak můžeme vidět na ukázce 2.2 [19]. **Weights & biases** může být využito k vizualizaci výsledků ladění. Ukázku můžeme vidět na obrázku 2.11 [20].

Listing 2.1 Ladění hyperparametrů ultralytics

```
from ultralytics import YOLO

# Initialize the YOLO model
model = YOLO('yolov8n.pt')

# Tune hyperparameters on COCO8 for 30 epochs
model.tune(data='coco8.yaml', epochs=30,
           iterations=300, optimizer='AdamW',
           plots=False, save=False, val=False)
```

Listing 2.2 Ladění hyperparametrů pomocí ray tune

```
from ultralytics import YOLO

# Load a YOLOv8n model
model = YOLO('yolov8n.pt')

# Start tuning hyperparameters for YOLOv8n training on the COCO8 dataset
result_grid = model.tune(data='coco8.yaml', use_ray=True)
```



Obrázek 2.11 Vizualizace ladění hyperparametrů pomocí weights and biases. Zdroj: [20]

2.6 Augmentace dat

Augmentace dat v sobě zahrnuje soubor metod úpravy zdrojových dat, jako je například změna odstínu, zrcadlové převrácení, pootočení, nebo tvorba mozaiky. Takováto úprava dat má velký vliv na schopnost modelu správně fungovat na nových datech. YOLOv8 při trénování provádí augmentaci dat automaticky a v této sekci přiblížím její fungování.

2.6.1 Odstín, sytost a saturace

Úprava těchto hodnot u trénovacích dat napomáhá modelu simulovat jiné světelné podmínky, než jsou v nich zachycené a model se jim tak lépe přizpůsobuje.

- **hsv_h** – upravuje hue, neboli odstín snímků
- **hsv_s** – upravuje saturaci snímků
- **hsv_v** – upravuje jas snímků

2.6.2 Rotace, posuny, škálování

Tyto transformace pomáhají modelu simulovat pohledy z jiných úhlů, pozic a vzdáleností.

- **degrees** – otáčí snímek o daný počet stupňů
- **translate** – provádí posun obrázku v závislosti na velikosti snímku
- **scale** – škáluje obrázek

2.6.3 Převracení a zrcadlení

- **flipud** – nastavuje pravděpodobnost, se kterou se obrázek otočí vzhůru nohama
- **fliplr** – nastavuje pravděpodobnost, se kterou se obrázek zobrazí zrcadlově obrácen

2.6.4 Výřezy a mazání

Vyříznutím či smazáním části obrázku nutíme, aby model vycházel pouze z toho, co mu zbývá, a to zlepšuje jeho schopnost generalizace.

- **erasing** – nastavuje, jak velká část obrázku bude smazána
- **crop_fraction** – nastavuje, jak velká část obrázku má být oříznuta z krajů

2.6.5 Mozaika a skládání augmentací

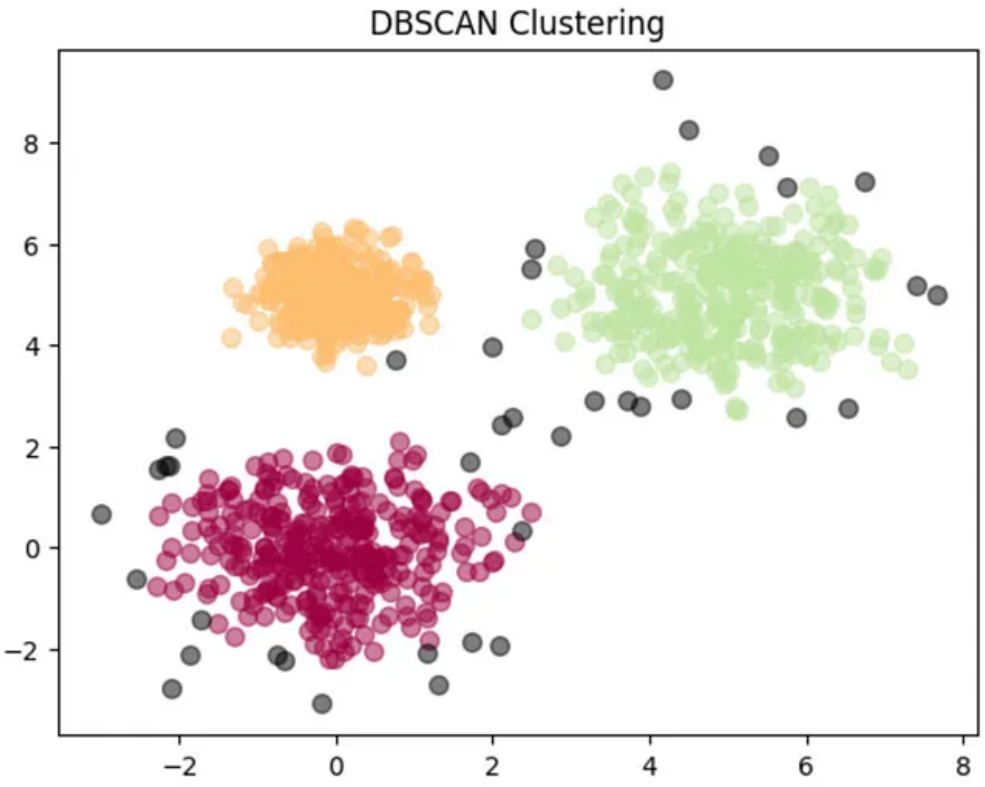
Jak jsem uvedl v sekci 2.3.2.2, YOLOv8 si z YOLOv4 převzalo mozaiku. Jedná se o způsob augmentace, ve kterém model skládá více trénovacích snímků do jednoho.

YOLOv8 je díky všem výše zmíněným metodám schopné tvořit komplexní trénovací data, která výrazně zvyšují připravenost modelu na nová, dosud neviděná data. Na obrázku 2.12 můžeme vidět příklad jedné dávky trénování.

2.7 DBSCAN

DBSCAN je shlukovacím algoritmem, který dobře pracuje s hustotou dat a umí odfiltrovávat odlehlé body. Funguje na bázi tvoření shluků pomocí klíčových bodů v datech. Pro každý datový bod je spočítán počet jeho sousedů ve vzdálenosti menší než **epsilon**. Jakmile má nějaký bod více než **min_samples** sousedů, stane se bodem klíčovým. DBSCAN pak bere tyto klíčové body a vytváří z nich clustery. Body, které jsou sousedy s klíčovým bodem jsou s ním ve stejném clusteru. Pokud spolu dva klíčové body sousedí, tvoří stejný cluster. Na obrázku 2.13 můžeme vidět tři clustery. Oranžový, zelený a červený. Šedé body v datech nespadají do žádného clusteru a jsou považovány za odchylky v datech. [21]

Obrázek 2.12 Trénovací dávka ukazující augmentovaná data. Zdroj: Vygenerováno při trénování modelu YOLOv8



Obrázek 2.13 Ukázka fungování DBSCANu. Zdroj: [22]

Kapitola 3
Analýza

V této kapitole popisuji, jak vypadají zdrojová data, jak jsou rozmanitá a kolik jich mám k disposici. Dále zde proberu možné postupy detekce a následného zpracování klíčových činností. V neposlední řadě také zmíním možnosti, jak zvyšovat spolehlivost našich výsledků. A nakonec zadefinuji metriky, kterými by se dala měřit úspěšnost celého řešení.

3.1 Analýza dat

Pro vytvoření dobrého řešení je nutno nejdříve pochopit, jak vypadají data, ze kterých budeme vycházet a jak bychom s nimi měli nakládat. V této sekci uvedu, kolik dat mám k disposici. Jak jsou rozmanitá a jak by se měla rozdělovat do trénovacích, validačních a testovacích množin.

3.1.1 Množství dat

Od firmy Profinit jsem k vypracování této bakalářské práce dostal k disposici data z japonského letiště Tokachi–Obihiro. Jedná se o celkem 313 videí o délce 15 minut. To jest celkem necelých 80 hodin záznamů. FPS neboli počet snímků za sekundu v těchto videích je 30. Celkový počet snímků v datasetu je $313 \cdot 15 \cdot 60 \cdot 30 = 8\,451\,000$. Toto číslo je velmi vysoké, ovšem většina těchto snímků je velmi podobná, protože se jedná o video, na kterém většinu času nedochází k velkým změnám.

3.1.2 Rozmanitost dat

Každý dataset je jiný a přináší s sebou svá specifika. V této subsekci se pokusím přiblížit, s jakými situacemi se můžeme v našich datech setkat a jaké obtíže s tím mohou být spojené. Rozeberu vliv denních dob a počasí a popíšu, jak vypadají letadla, která v datech přijíždí na letištní stojánku.

3.1.2.1 Počasí a denní doba

Odbavování letadel je zachycováno ve 4 různých podmínkách. Těmi jsou:

- slunečno – obrázek 3.1a
- zataženo – obrázek 3.1b
- deštivo – obrázek 3.1c
- noc – obrázek 3.1d

Náročnost detekce u slunečného a zataženého počasí se nijak dramaticky neliší. Složitější podmínky jsou v záznamech pořízených za deště a v noci.

Déšť mění vzhled letištní plochy. Za slunečného a zataženého počasí jsou čáry na letištní ploše dobře viditelné, ale za deště se mokrá plocha leskne, a tak nejsou důležité značky vidět – např. značka, na které má letadlo zastavit. Dalším faktorem zhoršujícím viditelnost jsou dešťové kapky pokrývající přímo objektiv kamery. Jedna taková kapka pak může zakrývat i větší část obrazu.

Záznamy pořízené v noci překvapivě nejsou problematické z hlediska viditelnosti. Letiště, na kterém jsou videa pořizována je velmi dobře osvětleno. Díky tomu jsou všechny pro nás důležité objekty bez problémů viditelné. Jediný problém je v tom, že objekty mohou vypadat odlišně než přes den. Například otevřené dveře zavazadlového prostoru jsou přes den výrazně tmavší než zbytek letadla. V noci ovšem je zavazadlový prostor osvícen, a tak vypadá obdobně světle jako ostatní části letadla.

(a) Ukázka slunečného počasí

(b) Ukázka zataženého počasí

(c) Ukázka deštivého počasí

(d) Ukázka stojánky v noci

Obrázek 3.1 Ukázky podmínek. Zdroj: Screenshoty z dat

3.1.2.2 Letadla

Na stojánku v datech přijíždějí dva typy letadel. Jedno je o poznání větší, příklad takového letadla lze vidět na obrázku 3.1a. Dveře jeho zavazadlového prostoru se otevírají vyklápěním směrem ven a tudíž jsou dobře viditelné. Tento fakt můžeme vidět na obrázcích 3.2a a 3.2b. Druhý typ letadla je menší, můžeme ho vidět na obrázku 3.1b. Jeho dveře se naopak vyklápějí dovnitř, z čehož vyplývá, že dveře při otevření nejsou viditelné, v letadle zůstane pouze patrný otvor, jak můžeme vidět na obrázcích 3.2c a 3.2d.

Většina letadel na videích je bílých a mají na sobě pouze logo **Japan Airlines**. V malé části videí je letadlo polepené. Problematičtější je to převážně, pokud jsou tyto polepy černé a poblíž dveří zavazadlového prostoru. V takovém případě jsou otevřené dveře hůře viditelné. Polepené dveře můžeme vidět na obrázku 3.3b a nepolepené na obrázku 3.3a.

(a) velké letadlo – zadní **(b)** velké letadlo – přední **(c)** malé letadlo – zadní **(d)** malé letadlo – přední

Obrázek 3.2 Porovnání vzhledu otevřených dveří letadel. Zdroj: Screenshoty z dat

(a) Dobře viditelné dveře **(b)** Hůře viditelné dveře kvůli polepům

Obrázek 3.3 Porovnání rozlišitelnosti dveří na bílém letadle vs na polepeném. Zdroj: Screenshoty z dat

3.2 Výběr a rozdělení dat

Data vychází z videa, které je rozděleno na jednotlivé snímky a ty jsou oanotované. Často se může stát, že dva snímky po sobě vzané jsou identické. Nedává tedy smysl dělit dataset obrázků z videí náhodně. Mohlo by se totiž stát, že v trénovací a validační množině se objeví prakticky stejný obrázek. Bude tedy lepší rozdělit data rovnou na úrovni videí a mít tedy trénovací, validační a testovací videa. Validační videa ještě rozdělím na dvě části. První část validačních videí bude testovat přesnost modelu při trénování. Na druhé části spočítám metriky a díky těmto výsledkům mohu dále ladit postprocessing modelu.

Každá sada dat by měla obsahovat všechny kombinace podmínek a letadel. Díky tomu bude model natrénován a vyhodnocen pro všechny nám dostupné podmínky.

3.3 Možnosti postupu

Pro vytvoření ideálního řešení je nutné nejprve vymyslet postup, kterým budeme dané problémy řešit. V této sekci budu popisovat různé možnosti, jak by bylo možné postupovat při řešení detekce klíčových činností. Zhodnotím použitelnost těchto postupů a zmíním jejich výhody a nevýhody.

3.3.1 Detekce zaparkování letadla

V následující subsekci se zaměřím na problém detekce zaparkování letadla a na způsoby, jakými by se dal řešit. Všechny možnosti se odvíjí od detekce předního kola letadla. Na zemi letištní stojánky je žlutým křížkem označeno místo, kde má letadlo po příjezdu zastavit svým předním kolem. Pozice předního kola na křížku je tedy nejlepším ukazatelem, že je letadlo zaparkované na správném místě.

3.3.1.1 Detekce křížku

Nejpřímočařejším postupem, jak zjistit, že letadlo zastavilo předním kolem na místě vyznačeném křížkem je detekce samotného křížku a předního kola letadla. Pokud jsou detekované pozice těchto dvou objektů blízko sebe, můžeme učinit závěr, že letadlo na křížku opravdu stojí. Jak jsem ovšem zmínil v sekci 3.1.2.1, značka na zemi při dešti není dobře vidět z důvodu odlesků na mokrém povrchu. Tato metoda je tedy závislá na počasí a není spolehlivá. Křížek je vidět na obrázku 3.4a. Ukázka počasí při kterém křížek vidět není je na obrázku 3.4b.

(a) Ukázky dobře viditelného křížku **(b)** Ukázky špatné viditelnosti křížku

Obrázek 3.4 Viditelnost křížků za různých podmínek. Zdroj: Screenshoty z dat

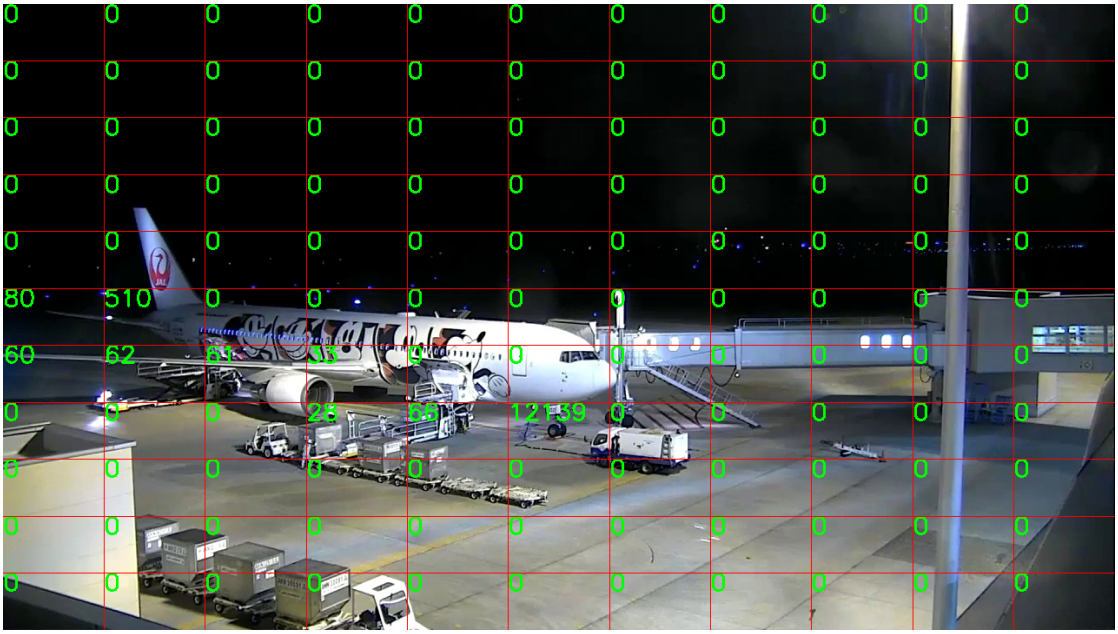
3.3.1.2 Průměrná pozice kola v datech

Dalším nápadem, jak řešit tento problém bylo využití průměrné pozice kola v datech. Jedná se o velmi jednoduchý přístup. Kolo stráví na parkovací pozici zdaleka nejvíce času. Z toho vyplývá, že množství dat, kde je kolo zaparkované je výrazně vyšší, než kdekoli jinde, a tak se průměr bude nacházet blízko parkovací pozice. Bohužel ale ve zbytku dat se letadlo a jeho kolo pohybuje směrem k ranveji a nikdy ne na opačný směr. Tento fakt vychyluje průměr částečně doleva nahoru, a není tak ideální reprezentací parkovacího místa. Vidět to můžeme na obrázku 3.5. Potřebujeme tedy vymyslet robustnější řešení.

Obrázek 3.5 Označení průměrné pozice kola v trénovacím datasetu. Zdroj: vlastní graf

3.3.1.3 Rozdělení obrazu na obdélníky

Toto řešení by mohlo spočívat v rozdělení obrazu na obdélníky. Mohl bych si pak uložit počet výskytů středu kola v datasetu v každém obdélníku a ten s největším počtem výskytů prohlásit za správné místo pro parkování letadla. Při správném nastavení počtu obdélníků pak jeden obdélník opravdu zahrnuje právě místo, kde by kolo letadla mělo zastavit při parkování. Problém je v nutnosti nastavování počtu těchto obdélníků tak, aby jeden obdélník dobře popisoval správné parkovací místo. Na našich datech to není problém. Máme data z jedné kamery a jednoho úhlu, takže stačí toto nastavit a detekce bude fungovat správně. Rád bych ale nalezl řešení, které bude obecně fungovat i při aplikaci na datech z jiného letiště, jiných kamer a úhlů pohledu. Model při takovémto přenosu bude potřebovat dotrénování, ale metody, které využívám na zpracování jeho výsledků by měly být přenosné. Ukázka obdélníků a četností výskytů kol v nich je na obrázku 3.6.



Obrázek 3.6 Obraz rozdělený na obdélníky. Čísla v obdélníku popisují počet výskytů kola v daném obdélníku ve videích z trénovacího datasetu. Zdroj: vlastní graf

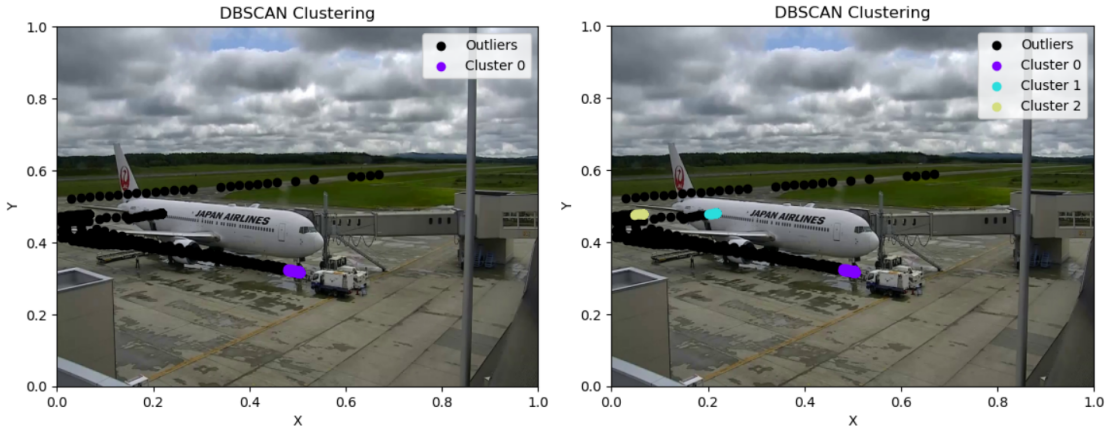
3.3.1.4 Clusterování pomocí DBSCANu

Přenosnou univerzálnější metodou by mohla být shluková analýza pozic kola v datech. Bude sice potřeba pro každou kameru na nových letištích shlukovat samostatně, ale takové řešení by mělo fungovat. V datech letadlo po většinu času stojí na vyznačeném místě. Vytváří se tam jeden velký shluk pozic, který se pokusím najít. Pokud bude pozice kola detekována v daném shluku, můžeme z toho odvodit, že letadlo zaparkovalo na správném místě. Na rozdíl od průměru je clustering robustnější, a tak najde opravdu nejčastější pozici kola v datech.

V našem případě lze vycházet z dat trénovacích. Není nutné zobecňovat pro různé pozice kamer a jejich úhly snímání. Pro větší obecnost je ale možné nechat model po dobu například jednoho dne detekovat pozice kola a použít takto získaná data ke clusteringu.

Ke clusterování budu používat algoritmus DBSCAN o kterém jsem psal v sekci 2.7. Pokud správně nastavíme parametry **eps** a **M**, můžeme dostat shlukování, které má jeden malý shluk okolo parkovacího bodu a zbytek jsou outliery. Když pak letadlo přijede na parkovací místo, kolo se dostane do parkovacího shluku a můžeme prohlásit, že je letadlo na místě.

Také je možnost nastavit M na nižší číslo a pak se nám v datech objeví shluky tři. Nejvíce bodů bude stále v parkovacím shluku, další shluky označují místa, kde letadlo stává po odbavení, když čeká na volnou ranvej. Tuto informaci by bylo možné také využít.



(a) 1 největší cluster **(b)** 3 největší clustery

Obrázek 3.7 Ukázka clusterování. Zdroj: vlastní graf

3.3.2 Detekce otevření zavazadlového prostoru

Vzhledem k tomu, že otevřené dveře zavazadlového prostoru vypadají úplně jinak, než dveře zavřené – ty nejsou od zbytku letadla rozeznatelné, stačí detekovat pouze stav otevřených dveří. Jakmile se dveře začnou otevírat, model to zaznamená a informaci předá k dalšímu zpracování.

3.3.3 Detekce připojení tunelu pro průchod pasažérů

V této subsekci zmíním možnosti detekce a následného odvození stavu připojení tunelu pro průchod pasažérů.

3.3.3.1 Detekce celého tunelu a letadla

První možností bylo detekovat celý tunel pro průchod pasažérů, u tohoto přístupu je ale komplikované, jak posoudit, kdy je opravdu tunel připojen a kdy je pouze poblíž letadla.

3.3.3.2 Detekce průniku hlavy tunelu a letadla

Druhou možností je detekovat pouze hlavu tunelu. Při tomto přístupu je jednodušší odvození, že tunel je opravdu připojen k letadlu. Například když bounding box letadla z 90 % překrývá hlavu tunelu, můžeme učinit závěr, že je tunel připojen. Pokud je hodnota překryvu nižší, usoudíme, že tunel se vzdálil od letadla.

3.3.3.3 Detekce stavu připojení

Třetí možností je detekovat pouze stav, kdy tunel je připojen. Při připojení k letadlu most spustí stříšku, jak můžeme vidět na obrázku 3.8d. Vypadá tak odlišně od mostu, který je sice blízko letadla, ale není připojen, viz obrázek 3.8c. Pokud letadlo není poblíž, můžeme se v datech setkat s mostem s vysunutou stříškou viditelnou na obrázku 3.8b a se stříškou zataženou, jak můžeme vidět na obrázku 3.8a. Při připojení mostu je tedy vytažená stříška a hlava nástupního mostu je z velké části zakrytá letadlem. Model by tak mohl být schopen detekovat přímo stav, kdy je tunel k letadlu připojen.

(a) Odpojený nástupní most bez vysunuté stříšky

(b) Odpojený nástupní most s vysunutou stříškou

(c) Připojující se nástupní most bez vysunuté stříšky

(d) Připojený nástupní most s vysunutou stříškou

Obrázek 3.8 Porovnání stavů nástupního mostu

3.4 Zvyšování spolehlivosti výsledků

V této sekci se zaměřím na cesty, jak zlepšit chování našich výsledků. Zmíním možnosti, jak vylepšit chování samotného modelu, popíšu důležitost timeoutu na potvrzení detekce a také rozeberu ladění pomocí nastavování minimální konfidence u detekce.

3.4.1 Zvyšování výkonnosti modelu

Zvýšení výkonnosti modelu je nejpřímější cesta, jak zlepšit chování celé aplikace. V této sekci popíšu cesty vedoucí ke zpřesnění detekcí modelu samotného. Chování modelu se dá zlepšovat poskytnutím většího a rozmanitějšího datasetu pro trénování, výběrem modelu s větším počtem parametrů a vyladěním hyperparametrů pro trénování modelu.

3.4.1.1 Zvyšování počtu anotovaných dat

Jednou z cest, jak zlepšovat chování modelu, je zvyšování velikosti datasetu. Model pak trénuje v rozmanitějších podmínkách, a to oddaluje přeučování. Jak jsem již zmiňoval výše, náš dataset aktuálně obsahuje 56 videí s délkou okolo hodiny. Čím více videí se mi podaří oanotovat, tím lépe bude model schopen reagovat na rozličné anomálie v datech. Problémem je, že anotace těchto videí je časově náročná a velké množství dat v datasetu také prodlužuje dobu potřebnou pro trénování.

3.4.1.2 Zvolení většího modelu

Výkon modelu můžeme jednoduše zvýšit využitím většího modelu s více parametry. Záleží na výpočetních prostředcích pro trénování a pro následnou detekci. Trénování většího modelu by na výkonných grafických kartách nebyl problém, ovšem pomalejší inference by mohla mít za následek nutnost větších časových kroků mezi detekcemi při real timovém použití. Častější detekce nám umožňuje ověřování výsledků detekce pomocí sady po sobě jdoucích detekovaných snímků.

3.4.1.3 Vyladění hyperparametrů

Vyladění hyperparametrů pro trénování modelu je výpočetně nejnáročnější možností, jak zlepšit chování výsledného modelu. O teorii ladění hyperparametrů a o jejich vlivech jsem psal v sekci 2.4. Problémem je, že počet kombinací hyperparametrů roste exponenciálně v závislosti na počtu hyperparametrů, které chceme ladit. Pro každou kombinaci musíme spustit trénování a ponechat jí dostatek času, aby se mohl vliv hyperparametrů projevit.

3.4.2 Timeout na potvrzení detekce

Při detekci modelu se může stávat, že určité objekty detekuje špatně. Takové chybné detekce ovšem často po pár snímkách zmizí. Je tedy možné počkat pár sekund a pokud daný objekt po celou tuto dobu detekujeme v nějakém stavu, můžeme potvrdit, že se objekt opravdu v daném stavu nachází.

3.4.3 Nastavení limitu konfidence

Nastavení limitu konfidence nám umožní jednoduše odfiltrovat detekce, u kterých si model není jist jejich správností. Zvýšením této hodnoty bychom dosáhli výrazného odfiltrování nechtěných chybných detekcí. Samozřejmě každé zlepšení jedné vlastnosti nese za následek zhoršení jiné.

Tento problém lze ale částečně vyřešit využitím class-specific konfidence thresholdu. Trik spočívá ve filtraci detekcí s nízkou konfidencí, kde ale tyto limity nastavujeme pro každou třídu rozdílně. Pokud tedy u nějaké třídy model často detekuje objekt s nízkou konfidencí, je možné limit této třídy snížit, a tím mu umožnit danou detekci učinit. Jiné třídy nemusí mít problém s nízkou konfidencí, jejich pravdivé detekce mohou být vyšší než 0,5, ale může docházet k halucinacím s konfidencí 0,3. Můžeme pak limit těchto tříd lehčích na detekci zvýšit, a tím odfiltrovat mylné detekce.

3.5 Návrh metrik pro hodnocení výsledků

Návrh metrik, které dobře popisují chování a spolehlivost modelu je zásadní pro dobré ladění řešení na validačních datech. Také je potřeba navrhnout metriky pro dobrou reprezentaci výsledků našeho řešení. V této sekci uvedu, jaké metriky budu užívat pro vyhodnocování fungování jak modelu samotného, tak i celého našeho řešení.

3.5.1 Metriky modelu

Při trénování modelu potřebujeme vědět, jak dobře se učí. V této sekci vysvětlím, jak fungují metriky používané pro vyhodnocování detekce objektů.

3.5.1.1 Intersection over Union

Intersection over Union neboli IoU nám dává způsob, jak vypočítat přesnost umístění bounding boxu. Vzorec pro IoU je definován takto: $IoU = \frac{\text{plocha průniku}}{\text{plocha sjednocení}} = \frac{A \cap B}{A \cup B}$. Ukázáno na obrázku 3.9

3.5.1.2 Precision, Recall a F1

Pro výpočet těchto hodnot potřebujeme nejprve zadefinovat co znamenají následující hodnoty:

- True Positive (TP) – detekujeme objekt, který se tam opravdu nachází
- True Negative (TN) – nedetekujeme nic, a žádný objekt se tam ani nenachází
- False Positive (FP) – detekujeme objekt, ale žádný tam není
- False Negative (FN) – nedetekujeme objekt, který ale na daném místě je

Na obrázku 3.10 můžeme vidět, co znamenají.

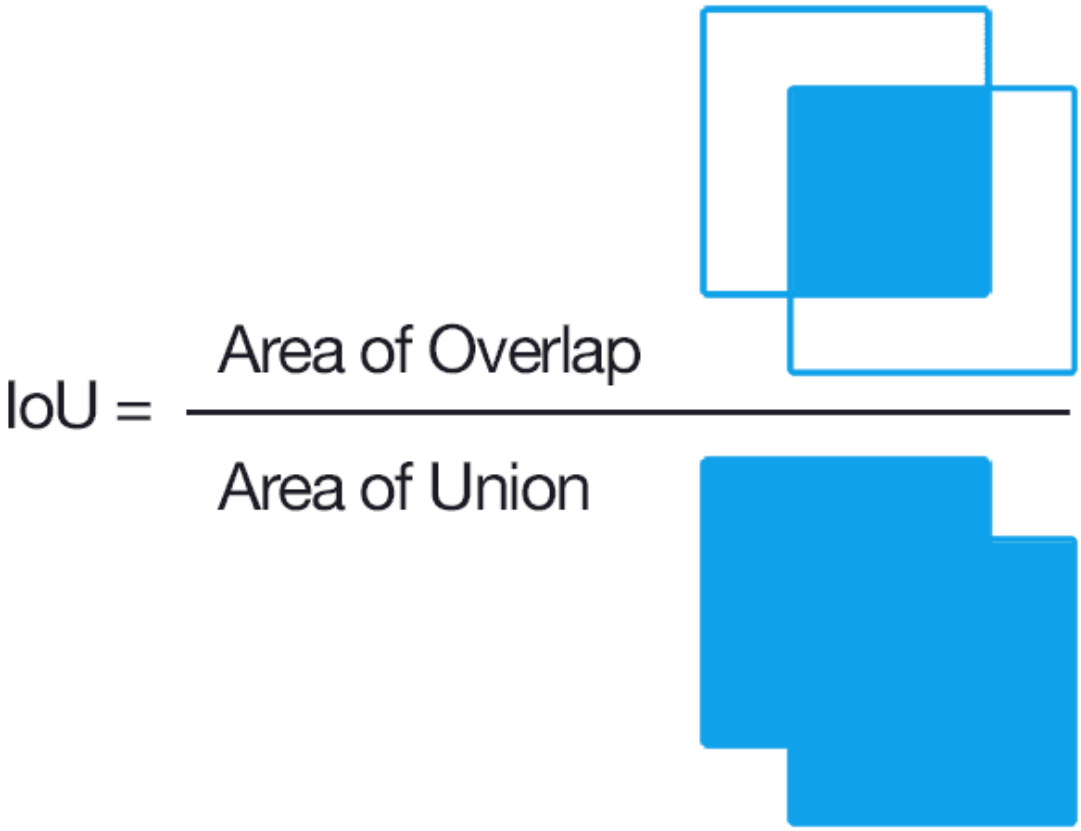
Precision definujeme jako podíl bounding boxů, které jsme detekovali správně oproti všem bounding boxům, které jsme detekovali.
$Precision = \frac{TP}{TP + FP}$

Recall je definován, jako podíl bounding boxů, které jsme detekovali správně oproti všem objektům, které jsme měli detekovat.
$Recall = \frac{TP}{TP + FN}$

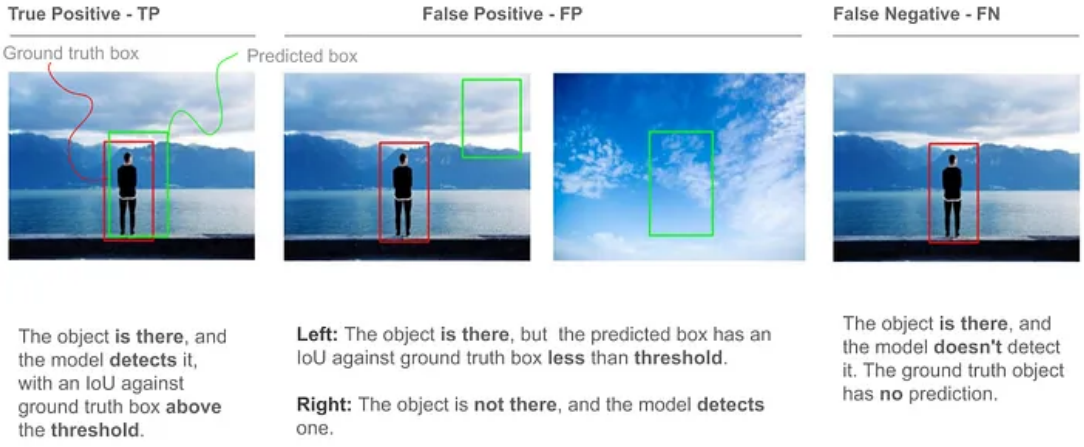
F1 skóre se používá ke spojení precision a recall do jedné hodnoty. Vzorec je harmonický průměr těchto dvou hodnot a vypadá takto:
$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

3.5.1.3 Precision-Recall Curve

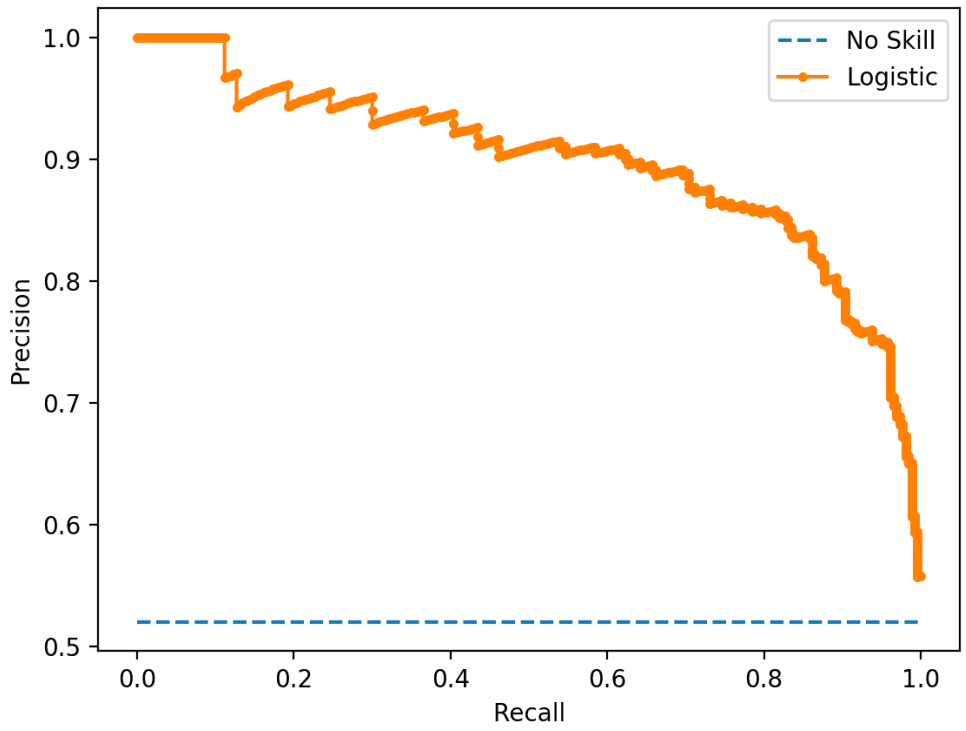
Tato křivka zobrazuje tradeoff hodnot precision a recall. Tyto hodnoty se vypočítají za použití různých konfidence thresholdů. Tento threshold udává minimální hodnotu konfidence detekce. Pokud je hodnota konfidence nižší, pak takovou detekci zahazujeme. Zahozením správné detekce nám klesne recall, pokud ale zahodíme špatnou detekci, stoupá nám precision. Vizualizaci takové křivky můžeme vidět na obrázku 3.11.



Obrázek 3.9 Ukázka výpočtu IoU. Zdroj: [23]



Obrázek 3.10 Vysvětlení TP, FP a FN. Zdroj: [24]



Obrázek 3.11 Křivka precison recall. Zdroj: [25]

3.5.1.4 Mean Average Precision

Average Precision neboli AP je definována jako plocha pod křivkou precision-recall. Tato hodnota je vypočítána pro každou třídu zvlášť a z těchto hodnot se vypočítá průměr. Tento průměr nazýváme mean Average Precision se zkratkou mAP. Tyto hodnoty jsou závislé na IoU thresholdu. Ten nám říká, jak přesně musí být umístěn bounding box oproti ground truth objektu.

Hodnota mAP50 nám dává informaci o mAP pro IoU threshold rovný 0,5. Často se používá mAP50-95, která dělá ještě průměr mAP pro hodnoty IoU mezi 0,5 a 0,95 v krocích po 0,05.

Pro naše řešení je dobrým ukazatelem metrika mAP50. Velká přesnost umístění bounding boxu pro nás není důležitá. Stačí nám vědět, zda klíčové objekty detekujeme, ale přesnost IoU 0,5 je pro nás naprosto dostačující.

3.5.2 Metriky řešení

3.5.2.1 Matice záměn

Jednou z metrik, kterou můžeme výsledky hodnotit, je použití matice záměn. Ta v sobě obsahuje 4 základní hodnoty. V následujícím rozpisu můžeme vidět tyto metriky a na příkladu je uvedeno, co pro nás znamenají:

- **True Positive (TP)** – detekujeme otevřené dveře a je to pravda
- **True Negative (TN)** – detekujeme dveře zavřené a je to pravda
- **False Positive (FP)** – detekujeme dveře otevřené, když jsou zavřené
- **False Negative (FN)** – detekujeme dveře zavřené, ale jsou otevřené

Na příkladu uvádím logiku ohledně otevření dveří. Stejně funguje metrika u stavu zaparkování letadla a připojení mostu.

3.5.2.2 Odchylky na krajích

Matice záměn nám dává dobrý vhled do obecného fungování našeho řešení, neřekne nám ale žádné bližší informace o tom, jak dobrý by byl náš model v praxi. Pro aplikování v reálném světě je důležité vědět, jak přesně model detekuje začátek a konec klíčových událostí. Bylo by tedy dobré měřit, jestli model detekuje otevření dveří, zaparkování letadla a připojení mostu o deset sekund dříve či později. To samé chceme měřit při zavření dveří, odjezdu letadla a odpojení mostu. Tyto hodnoty nám dají dobrý vhled do detailního chování modelu. Můžeme díky nim pak ladit zpracování detekcí modelu, a tak optimalizovat výsledky.

Pokud se stane, že na většině videí má řešení odchylku deseti sekund, ale na jednom videu s velmi špatnými podmínkami se nám detekce nedaří a výsledná odchylka je v řádu minut, bude lepší tyto případy rozdělit a prezentovat odděleně jako běžné chování řešení a chování řešení za špatných podmínek.

Popis hodnot v metrikách

- **true_start** – počet sekund od začátku videa do začátku klíčové činnosti v realitě
- **detected_start** – počet sekund od začátku videa do začátku detekce klíčové činnosti
- **true_end** – počet sekund od začátku videa do konce klíčové činnosti v realitě
- **detected_end** – počet sekund od začátku videa do konce detekce klíčové činnosti

Popis výpočtu odchylek Bereme v potaz pouze kladné hodnoty odchylek, proto nejmenší výsledek může být nula.

- **brzký začátek** – klíčová činnost byla zaznamenána dříve, než začala v realitě:
  $min(0, true\_start - detected\_start)$
- **pozdní začátek** – klíčová činnost byla zaznamenána později, než začala v realitě:
  $min(0, detected\_start - true\_start)$
- **brzký konec** – detekce klíčové činnosti byla ukončena dříve, než v realitě:
  $min(0, true\_end - detected\_end)$
- **pozdní konec** – detekce klíčové činnosti přetrvávala déle, než v realitě:
  $min(0, detected\_end - true\_end)$

3.5.2.3 Vyhodnocení chybných detekcí

Nejzásadnějším problémem, který může vyvstat je chybná detekce. Nejedná se pouze o pár sekund dřívější či pozdější detekce. Jedná se o detekování situace mimo chvíle, kdy k ní dochází a také o mezery v detekci v případech, kdy jsme stále měli klíčovou činnost detekovat. Pokud detekujeme, že letadlo odjelo ve chvíli, kdy stále stojí na stojánce, nastane v praxi chaos. Obsluha letiště počítá, že stojánka je volná a informuje jiné letadlo, že může zaparkovat na ní. Jiné letadlo přijede k plné stojánce a máme problém. Tyto situace musíme omezit na minimum. Chybné detekce se dají rozdělit do 3 kategorií. Těmi jsou:

- **brzká chybná detekce** – délka intervalu detekce, v případě, že začne a skončí před začátkem správného intervalu

  počet brzkých chybných detekcí – počet souvislých intervalů spadajících do kategorie brzké chybné detekce

- **mezera v detekci** – délka přerušení detekce, v případě, že celé přerušení spadá do intervalu klíčové činnosti v realitě

  počet mezer v detekci – počet souvislých intervalů spadajících do kategorie mezera v detekci

- **pozdní chybná detekce** – délka intervalu detekce, v případě, že začne a skončí po skončení správného intervalu

  počet pozdních chybných detekcí – počet souvislých intervalů spadajících do kategorie pozdní chybné detekce

U těchto situací bude dobré brát v potaz počet chybných detekcí i jejich délku. Není totiž takový problém, když tato chyba trvá pár sekund. Obsluha letiště by na takové hlášení nestihla zareagovat a k problému by tedy nedošlo. Samozřejmě pokud by k takovým kratším chybám docházelo často, nebyl by systém spolehlivý, a tím pádem by nebyl ani použitelný.

Kapitola 4
Implementace

V této kapitole budu využívat nástroje, metody a postupy zmíněné v předchozích kapitolách při samotné implementaci řešení. Od zpracování dat ze syrové podoby do formy použitelné pro trénování modelu přejdu k trénování samotnému. Zmíním, jak jsem porovnával použitelnost metod sepsaných v předchozí kapitole. Nakonec proberu implementaci výpočtu metrik zmíněných výše a pomocí informací, které mi tyto metriky dají na validačních datech, se ještě pokusím vylepšit chování řešení.

4.1 Data

V této sekci popíši, jak jsem nakládal s daty, které mám k disposici. Popíšu předzpracování dat a jejich převod na jednotné celky. Zmíním, jak jsem se rozhodl data rozdělit na trénovací, validační a testovací množiny. Popíši, jak jsem postupoval při anotaci dat a které objekty jsem labeloval.

4.1.1 Předzpracování dat

Jak jsem zmiňoval v subsekci 3.1.1, k disposici mám 313 videí o délce 15 minut. Na těchto videích jsou zaznamenány části odbavování letadla. Pospojoval jsem je do souvislých celků tak, aby jedno video zobrazovalo celkové odbavení jednoho letadla. Tato výsledná pospojovaná videa jsou dlouhá 45 min až 1 h 45 min, podle rychlosti obsluhy a náročnosti vykonávaných činností.

Při spojování videí jsem řešil na problém s velikostí spojených celků. Množství běžně dostupných nástrojů neumožňuje nastavovat bitrate. U našich dat je nízký. Pohybuje se mezi 200 a 500 kbps. Bitrate u videí s rozlišením 1280x720 bývá přes 1 000 a nástroje na spojování mají jako minimum takovou hodnotu. Po spojení videí se může stát, že z 12 GB dat budeme mít zbytečně 50 GB. Je tedy nutné zajistit podobný bitrate, jako u původního videa. Využil jsem nástroj HD video converter factory [26], který umožňuje ponechat bitrate původního videa. Výsledkem spojování bylo:

- 17 videí při zataženém počasí
- 14 videí natočených v noci
- 12 videí za deště
- 13 za slunečného počasí

4.1.2 Výběr a rozdělení dat

Trénovací i validační množiny musí obsahovat videa ze všech čtyřech podmínek. Minimálně tedy musím oanotovat 8 videí. To bude základ datasetu. Nejspíše bude nutné tato data rozšiřovat, protože kombinací situací v datech je mnoho. Zbylá nepoužitá videa budou sloužit pro testování našeho řešení. Budu klást důraz na striktní oddělení těchto datových množin.

4.1.3 Labeling dat

U labelingu dat je nutná konzistence. V této subsekci zmíním, jak budu data anotovat a vydefinuji pravidla pro anotaci každé klíčové činnosti, kterých se budu držet při anotaci dat. Videa z datasetu anotuji každou sekundu, to znamená, že z hodinového videa získám 3 600 anotovaných obrázků.

4.1.3.1 Labeling otevřených dveří zavazadlového prostoru

Velké letadlo Otevřené dveře zavazadlového prostoru labeluji od chvíle, kdy se u velkého letadla objeví dvě „okýnka“, která můžeme vidět na obrázku 4.1a. To je první znak, že se dveře otevírají. Labeling pokračuje po dobu procesu otevírání 4.1b, plného otevření 4.1c a zavírání 4.1b. Labeling ukončuji, jakmile se dveře plně zavřou a „okýnka“ přestanou být vidět 4.1d.

(a) dvě „okýnka“ **(b)** otevírání/zavírání **(c)** plně otevřené dveře **(d)** zavřené dveře

Obrázek 4.1 Ukázka stavů dveří velkého letadla

Malé letadlo U malého letadla labeluji od chvíle, kdy se objeví mezera mezi dveřmi a zbytkem trupu 4.2a. Dveře se následně otevřou plně 4.2b a na konci procesu nakládky a vykládky se zavřou. Za zavřené považuji dveře od chvíle, kdy se spojila se zbytkem trupu, a tak není na obraze patrná mezera. 4.2c

(a) otevírání/zavírání **(b)** plně otevřené dveře **(c)** zavřené dveře

Obrázek 4.2 Ukázka stavů dveří malého letadla

4.1.3.2 Labeling připojeného tunelu

Připojení tunelu labeluji po dobu, kdy je jeho stříška plně připojena k letadlu 4.3a. Anotaci ukončuji ve chvíli, kdy se stříška začne zvedat a je viditelná mezera mezi ní a trupem letadla 4.3b.

(a) plně připojen **(b)** připojování/odpojování **(c)** plně odpojen

Obrázek 4.3 Ukázka připojování tunelu

4.1.3.3 Labeling předního kola letadla

Přední kolo letadla v datech labeluji po celou dobu, kdy je dobře viditelné. Jako hranici pro „dobrou“ viditelnost jsem stanovil, že se kolo musí nacházet na světlejším povrchu letištní stojánky a ne na příjezdové cestě. Tato obraz je znázorněna vyšrafováním na obrázku 4.4. V noci anotaci provádím po celou dobu, kdy je kolo viditelné na obraze. Tento fakt je limitován kvalitou osvětlení letištní stojánky.

Obrázek 4.4 prostor ve kterém labeluji kolo. Zdroj: vlastní screenshot

4.1.4 Nastavení konfigurace datasetu

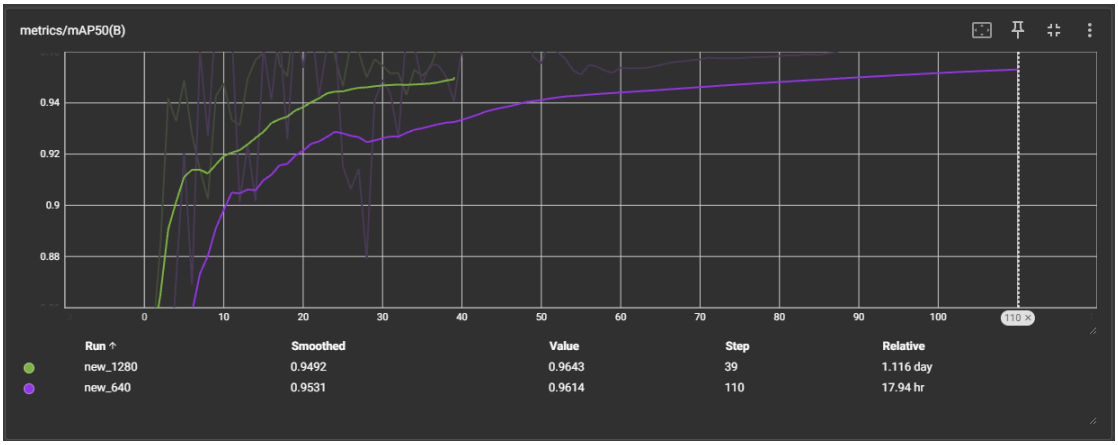
YOLOv8 pracuje s .yaml konfiguračními soubory k nastavení informací potřebných k trénování. Train a val hodnoty určují relativní cesty k trénovacím a validačním datům. Hodnota **nc** určuje počet tříd, které se snažíme detekovat a **names** obsahuje seznam názvů těchto tříd. Pořadí tříd musí korespondovat s očíslováním tříd v labelingu. Příklad .yaml souboru, který používám je k vidění v ukázce 4.1.

Listing 4.1 Ukázka .yaml konfigurace

```
train: airport_dataset_new_approach/train/images
val: airport_dataset_new_approach/val/images

nc: 3
names: ["Cargo_Door_Open", "Tunnel_Attached", "Wheel"]
```

4.2 Prvotní trénování modelu

Prvotní model trénoval na čtyřech videích, která zachycují všechny světelné podmínky. Další čtyři videa byla použita jako validační. Tato data byla anotována podle pravidel popsaných v sekci 4.1.3. Natrénoval jsem dva modely. Jeden s rozlišením 640 px a druhý s rozlišením 1280 px. Oba modely na čtyřech validačních videích vykazovaly dostatečně dobré výsledky. Menší model natrénovaný na zmenšených obrázcích měl problémy s detekcí kola letadla, a tak jsem zvolil model 1280 px. Na obrázku 4.5 jde vidět graf mAP50 těchto modelů v závislosti na počtu trénovacích epoch.



Obrázek 4.5 Porovnání MaP50 modelů. Zdroj: vlastní screenshot

4.3 Zpracování detekcí

V této sekci vyberu nejlepší kombinaci postupů zmíněných v kapitole analýza. Uvedu, jak budu postupovat při zpracovávaní detekování připojení tunelu, parkování letadla a otevření dveří zavazadlového prostoru.

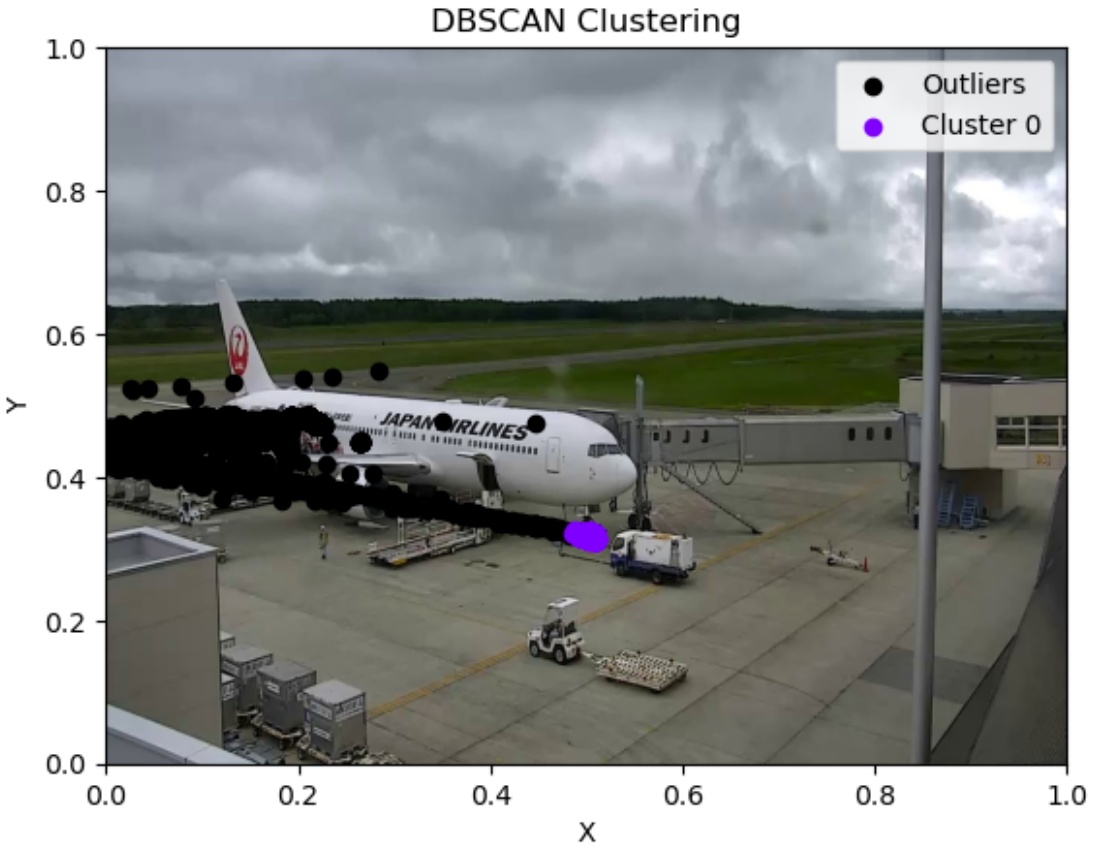
4.3.1 Tunel a dveře zavazadlového prostoru

Pro detekci připojení tunelu jsem se rozhodl použít metodu popsanou v sekci 3.3.3.3. Ta je, stejně jako metoda na detekování dveří zavazadlového prostoru, velmi přímočará. Jakmile je detekovaná její příslušná třída, zpracováváme informaci, že k dané klíčové činnosti dochází.

4.3.2 Letadlo parkuje na vyznačeném místě

Pro detekci stavu, kdy je letadlo zaparkované na správném místě a můžou na něm tedy být prováděny úkony spojené s přípravou na další odlet, používám přístup zmiňovaný v sekci 3.3.1.4.

Chci demonstrovat, že tento postup je přenosný i při přechodu na jinou kameru, na jiném místě, snímající pod jiným úhlem. Data o pozicích tedy neberu z anotované trénovací množiny, ale simuloval jsem, že kamera po nějakou dobu snímá pozice kol. Dal jsem tedy modelu k detekci pár videí z datasetu. Ukládal jsem pozice, které model detekoval a na takto získaných datech jsem provedl clustering. Parkovací místo z něj bez problému vyplynulo, a tak víme, že by tento postup měl být použitelný i na nových datech. Tento fakt můžeme vidět na obrázku 4.6



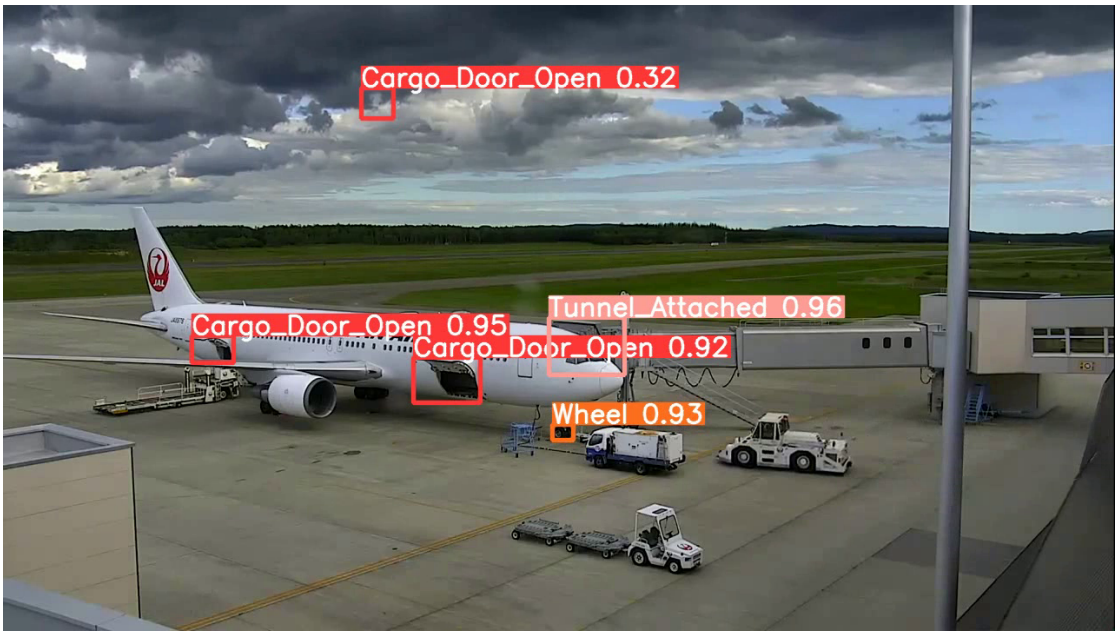
Obrázek 4.6 DBSCAN clusterování na datech z detekcí. Zdroj: vlastní screenshot

4.4 Zvyšování robustnosti v praxi

4.4.1 Timeout na potvrzení detekce

Informace z modelu jsem se rozhodl zpracovávat následujícím způsobem. Při první detekci nějaké události začne postprocessing počítat čas, po který je daná detekce aktivní. Pokud model detekuje událost nepřetržitě po dobu například 2 vteřin, je změněn stav. Například když model 2 s detekuje otevřené dveře letadla, vyvodíme že dveře jsou opravdu otevřené. Tento krok zvyšuje robustnost modelu. Ve výjimečných případech se totiž může stát, že model na jednom snímku udělá chybnou detekci. Taková detekce je zpravidla opravdu odchylkou a po pár snímkách zmizí. V datech se

mi povedlo v této chvíli nalézt jednu takovou chybu. Tou je detekce otevřených dveří nákladního prostoru v mracích, když obrys mraků připomínal tvar právě otevřených dveří. Chyba je vidět na obrázku 4.7.



Obrázek 4.7 Chyba modelu na validačních datech. Zdroj: vlastní screenshot

4.4.2 Nastavení limitu konfidence

Základní chování našeho modelu je takové, že při vizualizaci konfidence píše vpravo od názvu třídy detekce, a to číslem v intervalu 0 – 1. Jak můžeme vidět na již zmíněném obrázku 4.7, konfidence detekce nákladních dveřích v oblacích je 0,32. To je velmi nízká hodnota. Základní chování YOLA filtruje detekce s konfidencí nižší než 0.25. Tento limit můžeme zvýšit, a tím se zbavit chybných detekcí. V tomto případě bychom odfiltrovali detekce kola ve větší vzdálenosti. Detekce vzdáleného kola při odjezdu letadla ze stojánky můžeme vidět na obrázku 4.8. Bylo by dobré využít class specific konfidence threshold, zmíněný v kapitole 3.4.3. YOLOv8 nemá tuto funkci implementovanou, ale je možné si jí doprogramovat. Její implementace je triviálně jednoduchá viz ukázka kódu 4.2.

Dveře a tunel model detekuje s velmi vysokou konfidencí, pokud je detekce opravdu správná. Snížením minimální konfidence u kola letadla umožníme modelu detekovat kolo i na větší vzdálenost. Mylná detekce kola mimo označená místa nás nezajímá. Neovlivňuje totiž další vyhodnocování situací. Jediný problém by byl, pokud by kolo bylo mylně detekováno na parkovací pozici. Takovéto omyly model často nedělá (v datech jsem se s tím nesetkal), a tak upřednostníme detekování kola při zhoršených podmínkách i za cenu potenciálně častější chybné detekce.

Obrázek 4.8 Ukázka nízké konfidence u detekce kola ve vzdálenosti. Zdroj: vlastní screenshot

Listing 4.2 Filtrování detekcí pomocí class specific konfidence thresholdu

```
@staticmethod
def __filter_by_confidence(detection_result, confidence_thresholds: list):
    """
    Filters detection results to keep only those with confidence above the threshold.

    :param detection_result: The result of a detection.
    :param confidence_thresholds: List of minimal confidences for each class.
    :return: Filtered detection results.
    """
    confidences = detection_result.boxes.conf
    # convert from float to long to enable indexing
    class_indices = detection_result.boxes.cls.long()
    # Create a tensor of thresholds based on class indices
    thresholds = torch.tensor([confidence_thresholds[i] for i in class_indices],
                              device=confidences.device)
    # Create a mask where confidence is greater than threshold
    mask = confidences > thresholds
    # Filter boxes using the mask
    filtered_detections = copy.deepcopy(detection_result)
    filtered_detections.boxes = detection_result.boxes[mask]
    return filtered_detections
```

4.5 Zpracování videa

Zpracování videa je cílem celé této bakalářské práce. V této sekci vysvětlím, jak celé řešení funguje a jak nastavit parametry, pomocí kterých můžeme jednoduše upravovat chování našeho řešení. Popíšu, jak funguje vizualizace výsledků a co ze zpracovaného videa můžeme vyčíst. Nakonec přiblížím, jak funguje uložení zpracovaných dat ve formě intervalů ve formátu json.

4.5.1 Vizualizace informací

Dle mého úsudku je nejlepším způsobem, jak prezentovat výsledky řešení, použití snímků vrácených modelem a do levého horního rohu přidávat statusy klíčových činností. Tyto údaje jsou výsledkem zpracování informací získaných z modelu. Pro přehlednost jsem zvolil zelenou barvu písma, pokud k činnosti dochází a červenou, když ne. Na obrázku 4.9 můžeme vidět snapshot ze zpracovaného videa. Takto zpracované video je možné při detekování zobrazit. Po dokončení zpracování umožňuje analyzér uložení videa do složky outputs.

Obrázek 4.9 Ukázka zobrazení výsledků ve videu. Zdroj: vlastní screenshot

4.5.2 Uložení intervalů

Z analyzovaného videa ukládám intervaly, kdy ke klíčovým událostem docházelo. Ty následně mohu porovnávat s mnou vytvořenými pravdivými intervaly, a takto získávat metriky potřebné k ladění a následnému otestování mého řešení. Formát ukládání je zobrazen v ukázce 4.3. Může v něm být takto uloženo více videí, místo start a end jsou čísla, reprezentující začátky a konce detekovaných intervalů. Length značí délku videa. Všechny tyto hodnoty jsou ve vteřinách.

Listing 4.3 Ukázka formátování intervalů

```
"nazev_videa": {
    "statuses": {
        "Door(s) open": [["start", "end"], ["start2", "end2"]]
        "Parked": [["start", "end"]],
        "Tunnel attached": [["start", "end"]]
    },
    "length": 3600
}
```

4.5.3 Nastavení parametrů

4.5.3.1 Stride

Video zpracovávám snímek po snímku. Frekvenci, se kterou se berou z videa určuje parametr **STRIDE**. Ten říká, jak velké kroky mezi snímky děláme. V našich videích je počet snímků za sekundu 30. Nastavením parametru **STRIDE=30** docílíme toho, že budeme analyzovat jeden snímek za každou sekundu videa. Čím nižší je tato hodnota, tím častěji bereme snímky, a z toho důvodu budou výsledky lepší, ale analýza bude trvat déle.

4.5.3.2 Limity konfidence

Parametr confidence_thresholds nastavuje minimální konfidenci pro jednotlivé třídy. Umožňuje nám omezovat počet halucinací u jedné třídy a zároveň připustit detekování jiné třídy i s výrazně nižší konfidencí. Id tříd jsou následující:

- **0** – otevřené dveře zavazadlového prostoru
- **1** – připojení nástupního mostu
- **2** – přední kolo letadla

4.5.3.3 Timeouty

Úpravou hodnot ve slovníku na ukázce 4.4 můžeme měnit délku timeoutů na potvrzení konfidencí. Jeho fungování je popsáno v sekcích 3.4.2 a 4.4.1. Nastavením vyšších hodnot zvyšujeme robustnost řešení vůči halucinacím a výpadkům v detekci. Musíme ale počítat, že detekce činností bude o daný počet sekund zpožděna.

Listing 4.4 Ukázka formátu slovníku pro nastavení timeoutů

```
confirmation_delays = {"Parked": 3, "Door(s) open": 12, "Tunnel attached": 10}
```

4.6 Metriky

4.6.1 Výpočet metrik

Vzhledem k tomu, že data, která mám k disposici jsou pouze videa bez jakékoli další informace, musím si udělat i soubory, do kterých uložím informace o tom, co se ve videu děje. Rozhodl jsem se tato data ukládat do souboru csv. Každý řádek představuje video. První sloupec obsahuje název videa a dalších 6 sloupců představuje začátky a konce klíčových událostí.

Data ze souboru načtu a následně je porovnávám s událostmi, které zaznamenal model. Ukládám si data o tom, jaké chyby se model dopustil před začátkem reálné události (FP), v průběhu dané události (FN) a po jejím konci (FP). Všechny druhy chyb, co v datech mohou nastat zmiňuji v sekci 3.5.

4.6.2 Zpracování a analýza metrik

4.6.2.1 První funkční model

Metriky prvního modelu natrénovaného na 4 videích modelu můžeme vidět v tabulce 4.1. Data odchylek jsou v sekundách a počty sčítají množství výskytů chybných intervalů. Jedná se o průměr metrik měřených na 8 validačních videích. Využíval jsem timeout 5 sekund pro dveře a nástupní most a 1 sekundu pro parkování. Detekce parkování funguje perfektně. Horší jsou metriky u připojení tunelu a otevření dveří.

Tabulka 4.1 Průměr metrik na osmi validačních videích

Metrika	Dveře otevřené	Zaparkováno	Tunel připojen
Brzký začátek	-	-	-
Pozdní začátek	17.0	2.875	0.25
Brzký konec	35.375	0.875	2.375
Pozdní konec	1.375	4.5	5.625
Brzká chybná detekce	-	-	-
Počet brzkých chybných detekcí	-	-	-
Mezera v detekci	17.75	-	-
Počet mezer v detekci	1.5	-	-
Pozdní chybná detekce	-	-	1.25
Počet brzkých chybných detekcí	-	-	0.125

4.6.2.2 Analýza problémů

Po bližším prozkoumání podrobnějšího rozpisu metrik u všech videí jsem zjistil, že problematická jsou čtyři videa z osmi. Lepší videa jsou pro potřeby letiště naprosto dostačující, jejich metriky jsou vidět v tabulce 4.2.

Horší videa obsahují jednu chybnou desetivteřinovou detekci připojení tunelu po skončení odbavení letadla a jsou k vidění v tabulce 4.3. Vzhledem k tomu, že je problém ojedinělý a pouze na deset sekund, nezpůsobuje nám takové potíže. Větším problémem u horších videí je špatná schopnost detekce otevření dveří. Je vidět, že máme problém s detekcí stavu, kdy jsou otevřené. To znamená, že máme více false negative. Po prohlédnutí výstupního videa je vidět, že model má problém s detekcí dveří z důvodu potisku v jejich okolí. O možnosti, že by tento problém mohl nastat jsem se zmínil již v kapitole 3.1.2.2. Na obrázku 4.10a můžeme vidět, že model zadní dveře vůbec nedetekuje. Tato situace se u daného videa střídá s detekcí, která má ale nízkou

konfidenci, jak můžeme vidět na obrázku 4.10b. Na obou snímkách ale vidíme, že status otevření dveří stále hlásí, že jsou otevřené. To je díky timeoutu na potvrzení detekcí, který zajišťuje, že chvilková ztráta detekce otevření dveří se neprojeví na výstupu.

(a) Model nedetekuje zadní dveře kvůli polepu.

(b) Model zadní dveře detekuje, ale s malou konfidencí.

Obrázek 4.10 Model má problémy detekovat zadní dveře kvůli polepu.. Zdroj: Screenshoty z dat

Tabulka 4.2 Průměr metrik lepších videí.

Metrika	Dveře otevřené	Zaparkováno	Tunel připojen
Brzký začátek	-	2.5	0.5
Pozdní začátek	6.25	1.75	1.75
Brzký konec	-	-	-
Pozdní konec	2.5	4.0	5.25

Tabulka 4.3 Průměr metrik horších videí

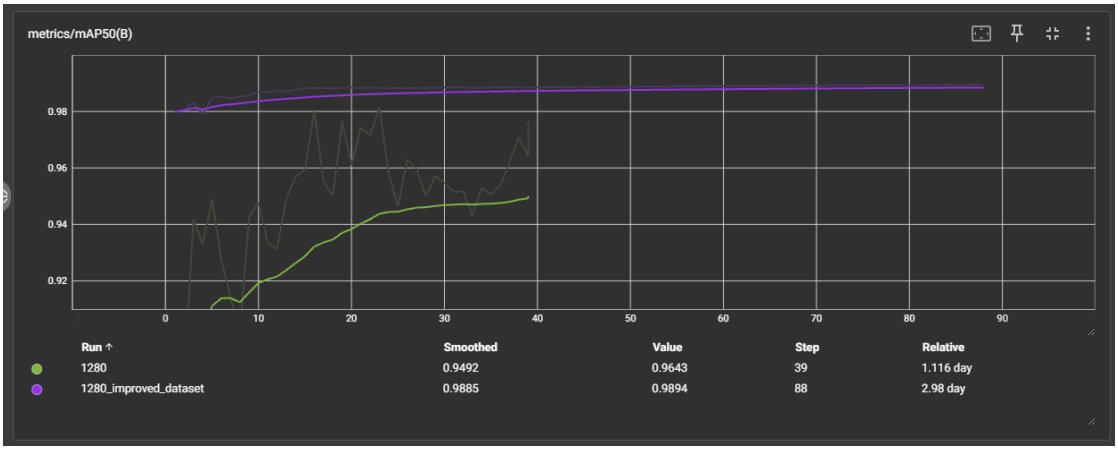
Metrika	Dveře otevřené	Zaparkováno	Tunel připojen
Brzký začátek	-	-	-
Pozdní začátek	27.75	3.25	2.5
Brzký konec	70.75	-	-
Pozdní konec	0.25	5.0	6.0
Brzká chybná detekce	-	-	-
Počet brzkých chybných detekcí	-	-	-
Mezera v detekci	35.5	-	-
Počet mezer v detekci	3.0	-	-
Pozdní chybná detekce	-	-	2.5
Počet brzkých chybných detekcí	-	-	0.25

4.6.2.3 Řešení problémů

Je vidět, že slabým článkem našeho řešení je samotný model. Prozatím trénoval pouze na čtyřech videích. Ta sice pokrývají všechny kategorie počasí a denních dob, ale už ne větší rozmanitost letadel a dalších možných odchylek. Proto jsem se rozhodl přidat do trénovací množiny další videa a tím dát modelu větší příležitost k natrénování detekcí na různorodějších datech.

Do trénovací množiny jsem přidal dalších pět videí, která zachycují obtížnější podmínky pro detekci. Tato videa v sobě obsahují náročnější situace, jako jsou polepené letadlo a vydatný déšť.

Na obrázku 4.11 můžeme vidět, že model trénovaný na vylepšeném datasetu byl už od prvních epoch výrazně lepší než předchozí.



Obrázek 4.11 Porovnání mAP50 starého modelu oproti vylepšenému. Zdroj: vlastní screenshot

4.6.2.4 Nastavování timeoutů

Vylepšený model na validačních datech měl uspokojivé výsledky, s výjimkou jednoho videa. Ve videu night_14 stále docházelo k mezerám v detekci. Tyto mezery můžeme vidět v úryvku 4.5. Jsou dlouhé čtyři, dvě, jednu, dvě a jedenáct sekund. Tyto intervaly jsou měřeny s nulovým timeoutem, takže přesně odpovídají detekcím modelu. Mezery jsou dostatečně krátké, takže použitím timeoutu je dokážeme překlenout. Největší mezera, je dlouhá jedenáct sekund.

Použitím timeoutu dvanácti sekund tedy plně řeší problém s vypadávající detekcí u dveří. U tunelu jsem nastavil timeout na deset sekund, abych předešel potenciálním halucinacím, nebo mezerám v detekci. Jedná se o preventivní opatření, na validačních datech neponese zlepšení. U parkování letadla jsem se rozhodl nastavit malý timeout třech sekund. Detekce zaparkování není tak náchylná na halucinace. Kolo letadla bychom museli mylně zdetekovat přímo na parkovacím místě, což je vysoce nepravděpodobné. Mylná detekce kdekoli jinde nijak neovlivňuje závěry řešení.

Timeouty způsobí opoždění detekcí, letiště ovšem měří čas po minutách a sekundy pro ně nejsou tak důležité. Timeout kolem deseti sekund tedy nijak neovlivní použitelnost našeho řešení v praxi. Timeouty jsou tedy nastaveny takto:

- **otevření dveří – 12 s**
- **připojený tunel – 10 s**
- **parkování letadla – 3 s**

Vliv těchto timeoutů na intervaly klíčových činností ve videu night_14 můžeme pozorovat v tomto úryvku 4.6. Je vidět, že každá klíčová činnost je popisována už jen jedním intervalem, který dobře odpovídá realitě.

Aplikoval jsem vylepšený model s takto nastavenými timeouty opět na všechna validační videa a dostal jsem následující metriky 4.4. Na validačních datech se nám tedy povedlo odstranit hrubé chyby na úkor zpožděných začátků a konců klíčových činností. Časové odchylky jsou do patnácti vteřin a ve většině videí jsou způsobené timeouty, se kterými můžeme počítat. Takto vyladěné řešení je finální a je možné ho otestovat na testovacích datech. Po aplikaci na testovacích datech již není možné jeho chování měnit.

Listing 4.5 Statusy ve videu night_14 bez timeoutu na potvrzení detekcí

```
"night_14": {
    "statuses": {
        "Door(s) open": [[182.0,197.0], [201.0, 203.0], [205.0, 2536.0],
            [2537.0, 2541.0], [2543.0, 2544.0], [2555.0, 2559.0]],
        "Parked": [[81.0, 2853.0]],
        "Tunnel attached": [[148.0, 2676.0]]
    },
    "length": 3600
}
```

Listing 4.6 Statusy ve videu night_14 s timeoutem na potvrzení detekcí

```
"night_14": {
    "statuses": {
        "Door(s) open": [[193.0,2570.0]],
        "Parked": [[81.0,2853.0]],
        "Tunnel attached": [[149.0,2677.0]]
        },
        "length": 3600
    },
```

Tabulka 4.4 Průměr metrik vylepšeného modelu s timeoutem na validačních videích.

Metrika	Dveře otevřené	Zaparkováno	Tunel připojen
Brzký začátek	-	4.5	-
Pozdní začátek	11.5	-	6.75
Brzký konec	2.5	-	-
Pozdní konec	8.875	5.0	10.25

Kapitola 5
Testování

V předchozí kapitole jsem popsal postup vytváření řešení detekce třech klíčových činností na letišti. Na validační množině jsem testoval jeho chování a podle toho přidával další videa do trénovací množiny pro model, ladil jsem timeout a konfidenční limity pro jednotlivé třídy. V této kapitole co nejlépe otestuji fungování vyladěného řešení na testovacích datech a zhodnotím použitelnost jeho výsledků. Použiji k tomu metriky vydefinované v sekci 3.5. Na testování používám dvanáct videí. U těchto videí využiji postup popsaný v sekci 4.6.1.

5.1 Průběh testování

Na testování jsem vybral dvanáct videí, tři pro každou kategorii světelných podmínek. Tato videa jsem prohlédl a označil jsem reálné intervaly, ve kterých dochází ke klíčovým činnostem. Nechal jsem naše řešení, aby prošlo každé video a také označilo intervaly, kdy ke klíčovým činnostem docházelo. Následně jsem pro každé video vypočítal odchylky a chyby, kterých se řešení dopustilo. Průměrné hodnoty metrik můžeme vidět v tabulce 5.1. Průměrné zpoždění začátku a konce detekcí u otevření dveří a připojeného tunelu je velmi podobné hodnotám timeoutu, který je pro tyto třídy nastaven. Takové zpoždění je tedy očekávané a značí, že naše řešení ve většině případů funguje přesně tak jak má.

Důležité ovšem je i vědět, jak se naše řešení chová v méně příznivých podmínkách. V tabulce 5.2 můžeme tedy vidět nejhorší chyby a odchylky, kterých se naše řešení daných dvanácti videích dopustilo.

Timeouty pro potvrzování detekcí jsem nastavil na **12 s** pro dveře, **10 s** pro tunel a **3 s** pro parkování. O ladění jsem psal v sekci 4.6.2.4. Při testování jsem nijak neměnil konfiguraci řešení.

Na obrázku 5.2d můžeme vidět matici záměn spojenou pro všechny klíčové činnosti. Jedná se o součet všech hodnot všech tří klíčových činností. Čísla na hlavní diagonále představují počet snímků, ve kterých došlo ke správnému určení situace. Vedlejší diagonála počítá snímky, kdy došlo k záměně. K detekci docházelo jednou za sekundu, takže jeden snímek odpovídá jedné sekundě záznamu.

5.2 Hodnocení výsledků jednotlivých klíčových činností

V této sekci zhodnotím výsledky chování detekcí každé klíčové činnosti. Proberu jejich chyby a odchylky a okomentuji, proč k nim dochází.

5.2.1 Otevírání dveří

Nejhorší hodnotou v metrikách je o čtyřicet jedna sekund opožděná detekce stavu otevření dveří. Toto zpoždění bylo způsobeno faktem, že náš model začal dveře s jistotou brát jako otevřené, až po jejich úplném otevření – obrázek 5.1c. Při otevírání se střídaly stavy bez detekce – obrázek 5.1a, se stavy, kdy model otevřené dveře sice detekoval, ale s nízkou konfidencí – obrázek 5.1b.

Třináct sekund dlouhá mezera v detekci měla podobnou příčinu, jako předchozí případ, ovšem problém nastal při zavírání dveří. Po celou dobu byly dveře otevřené a model to zaznamenával bez problému. Při zavírání na dobu delší než timeout přestal otevření dveří detekovat. Na posledních třináct sekund detekci obnovil a tím vznikla mezera. Následně se dveře zavřely, model to správně zaznamenal a detekci ukončil.

Obě dvě výše zmíněné situace nastaly na videích pořízených v noci zaznamenávající větší letadla. Detekce zaparkování dveří je u těchto případů spolehlivá po dobu, kdy jsou plně otevřené. Při otevírání a zavírání v nočních podmínkách je nutné počítat s menší spolehlivostí. Proces otevírání a zavírání dveří není delší než minuta. U menších letadel se jedná o pár sekund, proto s nimi nejsou problémy. Takové odchylky jsou pro letiště přípustné, vzhledem k tomu, že všechny časy se zaznamenávají v minutách.

Na obrázku 5.2a můžeme vidět matici záměn pro detekci otevření dveří. Ve zhruba 12 hodinách záznamu 117 sekund řešení mylně prohlašovalo, že jsou dveře otevřené, i když byly v realitě zavřené a 179 sekund byl závěr, že jsou zavřené, ale byly otevřené. Model se za dvanáct hodin záznamu mýlil tedy pouhých pět minut.

(a) nízká konfidence **(b)** bez detekce **(c)** plně otevřené – vysoká konfidence

Obrázek 5.1 Problémy s detekcí dveří při otevírání/zavírání, zdroj: vlastní screenshoty

5.2.2 Připojení tunelu

Detekce připojení tunelu je časově přesnější. Je to především díky tomu, že proces připojování tunelu je výrazně rychlejší. Z toho vyplývá, že doba po kterou si model není jistý je výrazně kratší. Samotné sklopení stříšky trvá kolem pěti vteřin.

Tunel za připojený považujeme až při plném připojení, na rozdíl od dveří. Ty považujeme za otevřené od počátku otevírání, až do plného zavření. Pro model je jednodušší detekovat toto úplné připojení, protože v daném stavu je výrazně déle, než v přechodném stavu připojování, a tak má model k disposici více trénovacích snímků.

Na obrázku 5.2b můžeme vidět matici záměn pro připojení tunelu. Za 12 hodin analýza celkem 112 sekund přehlížela připojenost tunelu a 121 sekund naopak detekovala, že je připojen i když nebyl. Tato čísla přibližně odpovídají deseti sekundové délce timeoutu vynásobené dvanácti videi. Detekce modelu na testovacích datech byly dostatečně přesné, že by takto vysoký timeout nebyl potřeba. Na validačních datech jsem ho nastavoval spíše preventivně, aby nedocházelo k halucinacím událostí z důvodu pár sekundové chybné detekce. Do budoucna by se ale nejspíše mohl snižovat.

5.2.3 Parkování

Nejlepší výsledky má detekce parkování. Nejhorší odchylka detekce začátku parkování byla tři sekundy. Detekce kola ve všech videích funguje bez problému, časové odchylky jsou zapříčiněné velikostí parkovacího místa. Jakmile se kolo letadla objeví v těchto mezích, spustí se timeout. Pokud se letadlo pohybuje pomalu, tak mu pár sekund trvá, než na daném místě zastaví. Rychlost příjezdu letadla tedy ovlivňuje tuto odchylku.

Odchylka odjezdu je ve všech případech opožděná. Kolo letadla musí totiž nejprve opustit parkovací místo a až poté se spustí timeout. Pokud je kolo letadla po dobu tří sekund detekováno mimo parkovací místo, analyzér prohlásí, že letadlo opustilo parkovací místo.

Ve všech videích se detekce parkování chová podle očekávání. Časová odchylka příjezdu je ±3 s. Detekce odjezdu je v průměru opožděná o 6,5 s. Nejhorší zpoždění je 9 sekund. Tento výsledek je ideální pro použití v praxi.

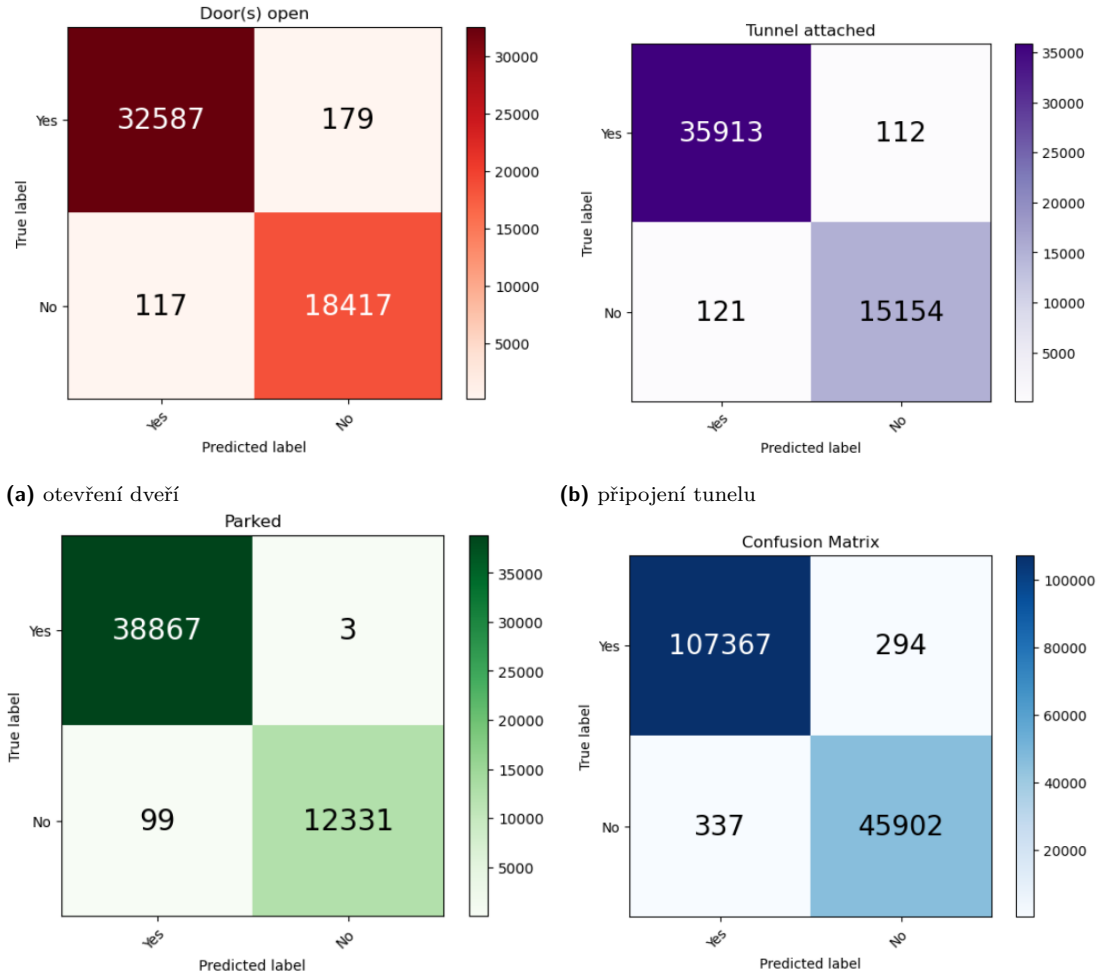
Na obrázku 5.2c můžeme vidět matici záměn pro zaparkování. Z této matice můžeme vyčíst, že model za celých 12 hodin jen 3 sekundy mylně prohlašoval, že letadlo není zaparkované, i když bylo. 99 sekund pak naopak hlásil, že letadlo parkuje, i když to nebyla pravda.

Tabulka 5.1 Průměr metrik finálního řešení na testovacích videích.

Metrika	Dveře otevřené	Zaparkováno	Tunel připojen
Brzký začátek	-	1.66	-
Pozdní začátek	12	0.25	9.33
Brzký konec	1.75	-	-
Pozdní konec	9.75	6.58	10.08
Mezera v detekci	1.083	-	-
Počet mezer v detekci	0.083	-	-

Tabulka 5.2 Nejhorší hodnoty metrik finálního řešení na testovacích videích.

Metrika	Dveře otevřené	Zaparkováno	Tunel připojen
Brzký začátek	-	3	-
Pozdní začátek	41	3	21
Brzký konec	21	-	-
Pozdní konec	21	9	18
Mezera v detekci	13	-	-
Počet mezer v detekci	1	-	-



(a) otevření dveří

(b) připojení tunelu

(c) parkování

(d) součet

Obrázek 5.2 Matice záměn pro jednotlivé klíčové činnosti, zdroj: vlastní grafy

Kapitola 6

Závěr

Cílem této bakalářské práce bylo co nejpřesněji zaznamenat parkování letadla, připojení nástupního mostu a otevření dveří zavazadlového prostoru, a to s co nejnižším počtem chybných detekcí.

Řešení prezentované v této bakalářské práci je schopné detekovat všechny tyto činnosti, a to s průměrnou přesností ±15 s. Nejvyšší zaznamenané zpoždění čítalo 41 s. Jedná se o velký posun, protože letiště, která nevyužívají řešení podobná našemu, zaznamenávají tyto události v řádech minut. K halucinacím činností dochází za běžných podmínek výjimečně. Modelu činí problém fungování za prudkého deště a větru, kdy kapky deště pokrývají velkou část objektivu, a s danými daty se nedá pracovat. Řešením tohoto problému může být vybavení kamer stěrači, které využívá firma Logipix.

Výrazné grafické úpravy letadel ztěžovaly modelu detekci. Model se rychle přeučoval na detekci dveří u čistě bílých letadel, protože ta v datech převažovala. Bylo potřeba v tomto ohledu zajistit rozmanitý dataset. Po trénování na rozšířeném datasetu již model bez problému prováděl detekce i na graficky upravených letadlech.

Řešení je připraveno na nasazení v reálném čase. Konvoluční neuronové sítě typu YOLO jsou schopny informace zpracovávat i na běžně dostupných zařízeních v real-time. Na procesoru Intel Core i5-11400 trvá inference kolem 200 ms. To znamená, že se dá dosáhnout 5 FPS. V bakalářské práci jsem prováděl veškeré testování s použitím jednoho snímku za sekundu, takže 5 snímků za sekundu je více než dostatečné. Provádění detekcí na grafické kartě Nvidia GeForce RTX 3060 Ti trvá kolem 10 ms. Jedna taková grafická karta je schopna zpracovávat záznamy z mnoha kamer současně.

Bibliografie

1. AIRLINES FOR AMERICA (A4A). *U.S. Passenger Carrier Delay Costs*. 2023. Available online: https://www.airlines.org/dataset/u-s-passenger-carrier-delay-costs/.
2. BORAH, Chinmoy. Evolution of Object Detection. *Medium*. 2022. Dostupné také z: https://medium.com/analytics-vidhya/evolution-of-object-detection-582259d2aa9b. [Accessed 28-03-2024].
3. BLAŠKO, Oliver. *Airport apron key object detection from surveillance cameras*. 2021. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology.
4. DING, Meng; DING, Yuan-yuan; WU, Xiao-zhou; WANG, Xu-hui; XU, Yu-bin. Action recognition of individuals on an airport apron based on tracking bounding boxes of the thermal infrared target. *Infrared Physics & Technology*. 2021, roč. 117, s. 103859. ISSN 1350-4495. Dostupné z DOI: https://doi.org/10.1016/j.infrared.2021.103859.
5. *Extended Object Tracking with Lidar for Airport Ground Surveillance - MATLAB & Simulink — mathworks.com* [https://www.mathworks.com/help/fusion/ug/extended-object-tracking-with-lidar-for-airport-ground-surveillance.html]. [B.r.]. [Accessed 05-05-2024].
6. ASSAIA. *Assaia Airport Technology* [https://www.airport-technology.com/contractors/airfield-safety/assaia/]. 2019. [Accessed 29-03-2024].
7. ASSAIA. *Insight: ApronAI Case Study: Reducing Kerosene Costs — assaia.com* [https://www.assaia.com/resources/reducing-kerosene-costs]. 2024. [Accessed 03-05-2024].
8. LOGIPIX. *Logipix Airside Augmented Reality Solution* [https://www.logipix.com/airside-augmented-reality/]. 2024. [Accessed 17-04-2024].
9. STUDIO, Label. *Open Source Data Labeling | Label Studio* [https://labelstud.io/]. 2024. [Accessed 29-03-2024].
10. BLOG, DagsHub. Best Object Detection Models. *DagsHub Blog*. 2024. Dostupné také z: https://dagshub.com/blog/best-object-detection-models/. [Accessed 28-03-2024].
11. BOESCH, Gaudenz. *A Guide to YOLOv8 in 2024* [https://viso.ai/deep-learning/yolov8-guide/]. 2024. [Accessed 09-04-2024].
12. HUSSAIN, Muhammad. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. *Machines*. 2023, roč. 11, č. 7. ISSN 2075-1702. Dostupné z DOI: 10.3390/machines11070677.
13. BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR*. 2020, roč. abs/2004.10934. Dostupné z arXiv: 2004.10934.

14. JOCHER, Glenn. *YOLOv5 by Ultralytics*. 2020-05. Ver. 7.0. Dostupné z DOI: 10.5281/zenodo.3908559.

15. WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. 2024.

16. RANGEKING. *Brief summary of YOLOv8 model structure*. 2023. Dostupné také z: https://github.com/ultralytics/ultralytics/issues/189. GitHub issue discussion in the Ultralytics repository.

17. DESAI, Chitra. Comparative Analysis of Optimizers in Deep Neural Networks. *International Journal of Innovative Science and Research Technology*. 2020, roč. 5, č. 10, s. 959–962. Dostupné také z: https://ijisrt.com/assets/upload/files/IJISRT20OCT608.pdf.

18. ULTRALYTICS. *Ray Tune* [https://docs.ultralytics.com/integrations/ray-tune]. 2024. [Accessed 24-04-2024].

19. LIAW, Richard; LIANG, Eric; NISHIHARA, Robert; MORITZ, Philipp; GONZALEZ, Joseph E; STOICA, Ion. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*. 2018.

20. ULTRALYTICS. *Weights & Biases* [https://docs.ultralytics.com/integrations/weights-biases/]. 2024. [Accessed 24-04-2024].

21. SCIKIT-LEARN DEVELOPERS. *DBSCAN clustering*. 2023. Dostupné také z: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html. Accessed: 2024-04-30.

22. YENIGÜN, Okan. *DBSCAN Demystified: Understanding How This Algorithm Works* [https://medium.com/@okanyenigun/dbscan-demystified-understanding-how-this-parameter-free-algorithm-works-89e03d7d7ab]. 2024. [Accessed 04-04-2024].

23. ROSEBROCK, Adrian. *Intersection Over Union (IoU) for Object Detection*. 2016. Available online: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

24. VEDOVELI, Henrique. Metrics Matter: A Deep Dive into Object Detection Evaluation. *Medium*. 2023. Dostupné také z: https://medium.com/@henriquevedoveli/metrics-matter-a-deep-dive-into-object-detection-evaluation-ef01385ec62.

25. BROWNLEE, Jason. How to Use ROC Curves and Precision-Recall Curves for Classification in Python. *Machine Learning Mastery*. 2023. Dostupné také z: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/.

26. FACTORY, Video converter. *HD Video Converter Factory Pro – Best Solution to Convert and Download HD Videos in Desired Format — videoconverterfactory.com* [https://www.videoconverterfactory.com/hd-video-converter/]. [B.r.]. [Accessed 29-04-2024].

Obsah příloh

```
Airport apron action detection
├── environment
│   └── environment.yml ..... konfigurace virtuálního prostředí ke spuštění BP
├── models
│   └── best.pt ..... nejlepší verze natrénovaného modelu
├── others
│   ├── cluster_centers_parking.csv ..... uložení parkovacího clusteru v csv
│   ├── wheel_occurrences.csv ..... uložení detekovaných pozic kola v csv
│   └── timestamps.csv ..... časy klíčových činností ve videích v csv
├── src
│   ├── metrics.py ..... metody pro výpočet metrik
│   └── video_analysis.py ..... nástroje pro analýzu videa
├── outputs
│   ├── intervals ..... složka ukládající intervaly z detekcí v json
│   └── videos ..... složka ukládající zpracovaná videa
├── raw_videos ..... složka ukládající videa ke zpracování
├── thesis
│   ├── thesis_source ..... složka se zdrojovými kódy práce ve formátu LaTeX
│   └── thesis.pdf ..... text práce ve formátu PDF
├── Detector.ipynb ..... jupyter notebook umožňující analýzu videí
├── Livestream.ipynb ..... jupyter notebook umožňující analýzu z livestreamu
├── MetricCalculator.ipynb ..... jupyter notebook počítající metriky řešení
└── README.md ..... popis repozitáře
```